Master's Thesis

FOUR FUNDAMENTAL SOFTWARE PROCESS MODELLING PRINCIPLES

THE CASE OF NOKIA TELECOMMUNICATIONS

Simo Rossi
Tero Sillander
13.05.1997

Department of Computer Science and Information Systems
University of Jyväskylä

ABSTRACT

Sillander Tero and Rossi Simo
Four Fundamental Software Process Modelling Principles – The Case of Nokia Telecommunications / Tero Sillander, Simo Rossi
Jyväskylä: University of Jyväskylä, 1997.
175 p.
Master's Thesis

It is a commonly accepted fact that the quality of software to a high degree relies on the quality of the software process which produces it. Therefore, the focus of software engineering has recently turned to study the software process and how it could best be improved. This change has led to the rise of a new discipline called software process modelling which aims at unambiguously representing the software process through explicit models built with process modelling languages and reasoning through those models.

For more than a decade software process modelling has received a great deal of attention in the field of software engineering, especially along the development of formal languages and process support tools. It has emerged to be a significant expedient in gaining understanding and improving software engineering processes. Software process modelling is used to facilitate understanding and communication of the software process, to support process improvement and management and to facilitate automatic guidance as well as to automate parts of the software process.

This study concentrates on the methodological aspects of software process modelling. First, the history and current state of software process modelling research are examined. Second, we introduce four fundamental software process modelling principles – issues to be considered when initiating software process improvement with modelling. These are deemed applicable for any modelling conducted in such a maidenly context with the aim to facilitate understanding and communication. Third, we experiment with the principles in an industrial software process modelling case and report the experience gained.

KEYWORDS: software process, software process modelling, software process model, software process modelling method, software process modelling method engineering, process modelling language.

If you don't know where you are, a map won't help.

– Watts S. Humphrey (1989)

TABLE OF CONTENTS

1 INTRODUCTION

Software engineering means a disciplined approach to the development of software. For over thirty years the importance of software engineering as well as the attention it receives have increased steadily. Software is used more widely now than ever and the intricate phenomena related to its development has been focused on in a variety of conferences and workshops. For example, eighteen International Conferences on Software Engineering (Proceedings of ICSE, 1996) and nine International Conferences on Software Engineering Education (Proceedings of ICSEE, 1996) have been held to date. Software quality has been discussed in already six International Conferences on Software Quality (Proceedings of ICSQ, 1996) and five European Conferences on Software Quality (Proceedings of ECSQ, 1996).

Notwithstanding, software engineering has barely progressed from the crisis it was in some thirty years ago: it is far from a disciplined and controlled area. In fact, the *software crisis* has been accepted to be more than a transitory characteristic prevailing in the software engineering domain. Agreeing upon Brooks' claim about there being no silver bullet to cure the crisis (Brooks, 1987), research emphasis is turning away from looking for a universal way to produce software. Instead, it has been recognised that multiple expedients are required to gain discipline to the field and to produce quality software.

Methods guiding and standardising the software development work and *CASE tools* (Computer Aided Software Engineering) supporting the methods are definitely among the most significant efforts to tame the software crisis. However, most of the method and CASE experience has found it difficult to adapt them to different development situations. Thus, *method engineering* has been introduced (Kumar and Welke, 1992) to mould the methods and various metaCASE and CAME (Computer Aided Method Engineering) tools have been developed to support it. Still, this increase in flexibility does not remove the methods' as well as the CASE tools' traditionally common concentration on only some of the software engineering activities. Current development has, however, begun to eliminate this deficiency (Harmsen, 1997).

In parallel with the methods development, another strand of research has produced *life-cycle models* dividing the software engineering activities into distinct stages to allow the process to be better managed. The first of them was developed already in the year 1956 (Benington, 1956). These abstract models are used mostly as management tools, but they do not qualify as guides for the actual software development (Curtis et al., 1992). Recently, it has been found that their support for the management is largely defective as well (Curtis et al., 1992, p. 75).

Shortcomings in the management of the software development process lead to the *process movement*, which emerged in the mid-eighties (Curtis et al., 1992), indicating a shift of research focus from the software product to the development process. This *software process* is defined as the sequence of the steps required to develop or maintain software (Humphrey, 1995, p. 4). Currently, it is commonly accepted that the quality of a software product to a high degree relies on the quality of the software process (Heineman et al., 1994; Humphrey, 1989). Accordingly, this subject has been focused on in four International Conferences on the Software Process (Proceedings of ICSP, 1996), five European Workshops on Software Process Technology (Proceedings of EWSPT, 1996) and ten International Software Process Workshops (Proceedings of ISPW, 1996).

The shift in focus is embodied in several *assessment frameworks* and *improvement approaches* such as the Capability Maturity Model (CMM) (Paulk et al., 1993) and the Quality Improvement Paradigm (Basili and Rombach, 1987; 1988) which aim at continuous software process assessment and improvement.

Process improvement is facilitated by process modelling among other things such as process evolution and process monitoring (Ellmer, 1995, p. 82). *Software process modelling* explicitly focuses on phenomena that occur during software creation and evolution, a domain different from that usually modelled in human-machine integration of information systems design (Curtis et al., 1992). This domain depicts the software process in *process models*, which are abstract descriptions of an actual or proposed process that represent selected process information that are considered important to the purpose of the model and can be enacted i.e. executed by a human or machine (Curtis et al., 1992). Software process modelling has been dealt with in many of the previously mentioned conferences and workshops, and even one modelling-dedicated workshop has been held (Proceeding of EWSPM, 1991).

The models can serve several purposes. According to Curtis et al. (1992), they can be used to facilitate understanding and communication of the modelled process, support process improvement and management, provide automatic guidance and automate parts of the software process. All of these serve process improvement, but one model cannot fulfil them all. Therefore, one should be selected.

Until now, software process modelling research has been strongly *process programming* oriented. Following Osterweil's (1987) example, the efforts have aimed at automating the process. A variety of process-centred software engineering environments (PSEEs) have been introduced to define, evolve, analyse, and execute process models. Still, understanding of the software process is needed as a foundation for all the other objectives.

This selection of the modelling objective provides the basis to select a suitable process modelling language (PML) as well. In addition to process technology, PMLs have been the main focus of the software process modelling research. Due to the programming-orientation of the modelling field, most languages are formal and thus serve best the automation objectives. They can be used to depict various perspectives of the software process, offering different points of view to the same process. The most commonly used perspectives are functional, behavioural, organisational and informational. Generally, the PMLs combine two or more of these to provide a complete process model.

Focusing on programming the software processes, the software process modelling community has left the methodological aspects of process modelling with less attention (Kontio, 1995, p. 32). Still, the need for flexible and modifiable methods and tools is evident not only in the software engineering field, but in the software process modelling domain as well. Apart from the implicit and superficial modelling methods incorporated in many PSEEs, only some non-technology oriented software process modelling methods have been devised to particularly support the complex task of the creation of software process models. These include Elicit (Höltje et al. 1994), the Descriptive Modelling Process (DMP) (Kellner and Hansen, 1989; Kellner and Hansen 1988), the Process Description Approach (PDA) (Klingler and Schwarting, 1994) as well as a modelling process introduced by Kontio (1995).

The assets of these methods comprise taking due regard of the process context and process data collection. Additionally, almost all of them select the PML dynamically, based on the modelling context and objectives. On the other hand, their description of the modelling process is rudimentary and in a rather general level of detail. Furthermore, they do not take into account the need to modify the selected PML(s) or the modelling method itself. Therefore, software process modelling method engineering including PML engineering (Koskinen, 1996; Koskinen and Marttiin 1997) is required. As was the case with software engineering methods, software process modelling methods and languages require method engineering to facilitate flexibility as well.

Purpose of the study. Related to the software process modelling method research, the purpose of this study is to examine factors which are crucial for successful software process modelling. Here we strongly emphasise the methodological aspects of process capture, leaving its technological facets out of the scope. Modelling success denotes the degree to which the produced models and the modelling itself fulfil the needs of the modelling environment. This broad research aim is narrowed with two presuppositions: the software process has not been formerly modelled on a detailed level and the modelling objective is to facilitate understanding and communication of the process.

Combining the assets of the selected software process modelling methods and complementing their shortcomings, four fundamental software process modelling principles are outlined. To explore the issue in practice, we apply these in a modelling process which is executed in an industrial software process modelling case.

Research questions. In conducting this investigation, the following problems are considered important.

1. What are software engineering and software process modelling and what is their current status?

2. How is software process modelling currently supported by modelling methods?

3. Which issues should be taken into account when modelling a software process for the first time to facilitate understanding and communication?

4. How should these issues be applied in practice?

Results. Eliciting the answers for the questions above yields two main contributions. First, the study submits four fundamental software process modelling principles: flexibility, multiple data sources, PML engineering and participation of process actors. These we find most essential for modelling with the presuppositions outlined above. The first two are generally supported in the examined software process modelling methods, but the third, PML engineering, has been completely neglected and the fourth, participation, has received very little attention. Second, our research puts forward a detailed experience report of an industrial software process modelling case. Such experience is necessary to further develop the theories of the modelling domain. It also strengthens the principles by shedding light on their applicability and limitations.

The results of this study recognisably suggest that the four modelling principles, provide software process modelling with excellent guidelines when the modelling is to facilitate understanding and communication in a maidenly context.

Research strategies and content of this report. Our effort can be considered via applying the multi-methodological systems development approach to information systems research (Nunamaker's et al., 1991). In the following, we briefly survey our study and the structure of this report, referring to the four strategies Nunamaker et al. (1991, pp. 94-96) list: theory building, observation, experimentation and systems development. This presentation illustrates one plausible adaptation of the research strategies in our work; other applications are also feasible.

The next three chapters present the basic terminology, the theoretical background and the current state of the research in the field of software process modelling. To begin with, Chapter 2 is aimed to give a superficial overview of software engineering. Then, Chapter 3 examines more closely the phenomena of software process modelling. Finally, Chapter 4 concentrates particularly on method driven software process modelling, leading us to the very core of our research.

After surveying the background literature, this document proposes four modelling principles as guidelines to be followed in software process modelling and outlines a modelling life-cycle applying the principles in practice (Chapter 5). Contributing to the knowledge in the research domain, this principle construction is an application of the *theory building* research strategy. The modelling life-cycle presented, again, represents an application of the *systems development* research strategy.

Chapter 6 presents the experience gained while executing the modelling life-cycle of the preceding chapter in an industrial software process modelling case. This depiction is rather detailed to adequately submit even the minor notices made. This part of our work is deemed to employ the *experimentation* and *observation* strategies in that it experimentally executes the modelling process of the previous and observes the context for preconditions and effects.

Then, looking towards the end of this report, Chapter 7 summarises the modelling experience and exhibits the lessons learned during the practical work. Finally, conclusions are discussed and issues for future work are suggested in Chapter 8.

2 SOFTWARE PROCESS MODELLING BACKGROUND

The purpose of this chapter is to outline the root of software process modelling. We begin with a brief presentation of software engineering (section 2.1), which recently has ramified into a multitude of research fields, including the study of software processes. Next, the course of our examination passes through software quality (section 2.2) and software process movement (section 2.6), leading to software process assessment and improvement (section 2.7 Software Process Assessment Frameworks and Improvement Approaches), which is seen as the impetus for the modelling itself.

This chapter is intended to provide a historical review covering the essence of software engineering progresses from a process modelling point of view. First, the nature of software engineering and the software crisis are described (Section 2.1). Second, the goal of most software engineering research efforts, software quality, is presented (Section 2.2). Third, the history and current status of the research is outlined, covering software engineering methods (Section 2.3) and method engineering (Section 2.4), life-cycle models (Section 2.5), software process movement (Section 2.6) and assessment frameworks and improvement approaches (Section 2.7). Finally, Section 2.8 summarises the yield of software engineering research.

2.1 Software Engineering

Software engineering means a disciplined approach to the development of software. The term is used to denote both the industrial domain and the field of science. Another concept closely related to software engineering is *software crisis*, which refers to the continued failure to properly manage software development (see for example Nutt, 1995, p. 324). During the last three decades, software engineering has received increasing attention in both the academic and the industrial territory because increased utilisation of the software in every day life (see the previous chapter). This owes to the inherent possibilities of the field and the prevalence of software crisis in it. As the demand for software is increasing, the pressure for achieving controlled production of quality software rises continuously. This subsection examines the nature of software engineering to shed light on the components of software crisis, which characterises the field today.

Distinctions compared to traditional engineering. Reviewing the literature of the software engineering field, the difficulty of managing software development is evident. The development of software differs significantly from the more traditional fields of engineering, which are far more disciplined. Kontio (1994, p. 13) lists minimal manufacturing costs, youth of the field and abstractness of software as the prime distinctions. Pressman (1994, p. 10) also remarks that software is a logical rather than physical system element, i.e., software is non-tangible, and Osterweil (1987, p. 3) speaks of a pure information product.

Software craft. Referring to the prevailing intuitive development methods, Watts Humphrey (1995, p. 2) even argues that the current practice of software engineering is closer to a craft than an engineering discipline. Lack of discipline is often attributed to some of the characteristics of the software development process. Below, we list some exemplary depictions of these problematic features.

Process features. Armenise et al. (1993, p. 402) characterise the software production process as a multiperson, largely intellectual design activity, in which co-operation among humans and human interaction play a fundamental role. Kumar and Welke (1992) mention its complexity and ill-structuredness[1]. Curtis et al. (1988, p. 1270) list the thin spread of application domain knowledge, fluctuating and conflicting requirements and communication bottlenecks as the three most salient problems.

Demand. Despite difficulties on the supply side, the demand for software products is continuously expanding. Kontio (1994, p. 13) illustratively portrays the uses of software in the modern society. Software is everywhere. Currently, especially the use of software in embedded systems seeks for its limits. Furthermore, the tendency is that everyone

[1] Kumar and Welke speak of information systems development, which covers software development as a subarea. For the purposes of this study, we make no distinction between information systems development and software development.

needs "more better software cheaper and faster" (Humphrey, 1995, p. 2). Consequently, the process and product requirements are highly unstable rendering the development process evolutionary and thereby even more cumbersome (Armenise et al., 1993, p 402).

Research. Despite intense research in the field, no cure for the crisis of software engineering has been found. Brooks (1987) stated the poor yield of the research in the field. His argument was that there is no single technique applicable to all software development situations, i.e. there is no silver bullet. The following years have brought no improvement to that state of affairs (Humphrey, 1989; Armenise et al. 1993, p. 402; Basili et al. 1994, p. 3; Kehoe and Jarvis, 1995, p. 13; Humphrey, 1995, p. 2)

To be brief, the rapid evolution of technologies and methods, increased complexity of the applications to be developed and human-orientation as well as lengthy duration of the processes (Conradi et al., 1993, p. 26) are the main reasons why software engineering remains evolutionary and why we can do nothing but "bite the silver bullet"[1].

The ultimate goal of most efforts dealing with the software crisis is the development of quality software. Traditionally, there have been two major strands of research striving for this objective. First, there is method oriented research having focused primarily on software products and their creation. Second, process oriented research has mainly concentrated on the software engineering process, i.e., the activities needed to create a software product. The rest of this chapter will deal with these research streams, which recently have found common ground and started to intertwine. First, however, the software engineering is reviewed from a business process perspective and the concept of software quality is defined.

---

[1] Harel (1992) uses the expression to denote something painful but necessary you have to do, or to undertake an activity despite critisism or opposition, while exhibiting a measure of courage and optimism.

2.2 Software Business Process and Software Quality

From a business point of view, a company's software process is a business process among others (such as the customer service process, the warehouse managing process, the order processing process etc.). Thus, software development is subject to the same improvement pressures as its fellow business processes. These pressures may comprise productivity, process cycle time (see Aoyama, 1993, p. 46), process consistency and predictability, business prospects and process management.

Nonetheless, "the primary concern of the software engineering community is the development and support of quality software." (Heineman et al. 1994, p. 501). This holds even though intense competition in the software markets has influenced a rise of other previously mentioned improvement areas (Kontio, 1995, p. 41-43). Accordingly, software quality has been focused in already six International Conferences on Software Quality (Proceedings of ICSQ, 1996) and five European Conferences on Software Quality (Proceedings of ECSG, 1996).

Focusing on quality does not exclude the other improvement objectives either. On the contrary, improving software quality leads to improved productivity as well, as is indicated by (Pohl, 1996, p. 49): "[...] the production of higher quality avoids cost and time-consuming rework and therefore leads to higher productivity."

The definition of quality is ambiguous since it depends on the standpoint of the observer. A definition of software quality is even more intricate to accomplish than a definition of the quality of a more conventional, tangible product. According to IEEE, *Software quality* is (see Jarke and Pohl, 1992, p. 346):

1. The totality of features and characteristics of a software product that bear on its ability to satisfy given need.

2. The degree to which software possesses a desired combination of attributes.

3. The degree to which a customer or user perceives that software meets his or her composite expectation.

4. The composite characteristics of software that determine the degree to which the software in use will meet the expectations of the customer.

To conclude, software quality is a relative notion, which cannot be measured in absolute terms.

Next, the path of our examination leads to investigating the more product oriented research stream, which has produced a variety of software engineering methods to guide the software development work.

2.3 Software Engineering Methods

A *method*[1] is a codified set of procedures the purpose of which is to guide the work and co-operation of the various parties involved in the development of software[2] (Hirschheim and Klein, 1992, p. 296). The first methods were merely documented "rules of thumb" passing on the software engineering experience. Later, features were borrowed from structured programming languages and recently methods have been filled with object orientation.

Structured methods, such as Yourdon's Modern Structured Analysis (Yourdon, 1989), are based on structured programming languages, whereas object oriented methods, such as Rumbaugh's famous OMT (Rumbaugh, 1991), are founded on object oriented programming languages (Coad and Yourdon, 1991a, p. 5). Coad's and Yourdon's OOA/OOD (Coad and Yourdon, 1991a, 1991b), again, are an examples of the broadest category of all methods today. Namely, they mix structured and object-oriented approaches. Object-orientation combines the data and its exclusive processing, which traditionally have been treated separately in structured methods.

[1] Hirschheim and Klein use the term methodology. In this study, however, we make no distinction between the two terms, but for unity's sake, we speak of method.

[2] Hirschheim and Klein (1992) use the term information systems development.

These methods and the like support the human problem-solving process, which involves concurrent analyses of both high-level and very detailed issues (Blum, 1994, p. 85). They are aimed at certain parts of the software's life-cycle and focus primarily to describe the software engineering products in various diagrams, tables, matrices and also with natural language. Many of them also include some form of a process description of their use as well. Recently, some methods have emerged which take into account larger parts of the development process (see Harmsen, 1997, p. 5).

Figure 1 below illustrates an example of the traditional relationship of software engineering (SE) methods to the software development. It represents a method describing a portion of a software process, the design phase, including steps to take and end products, software designs, to produce.

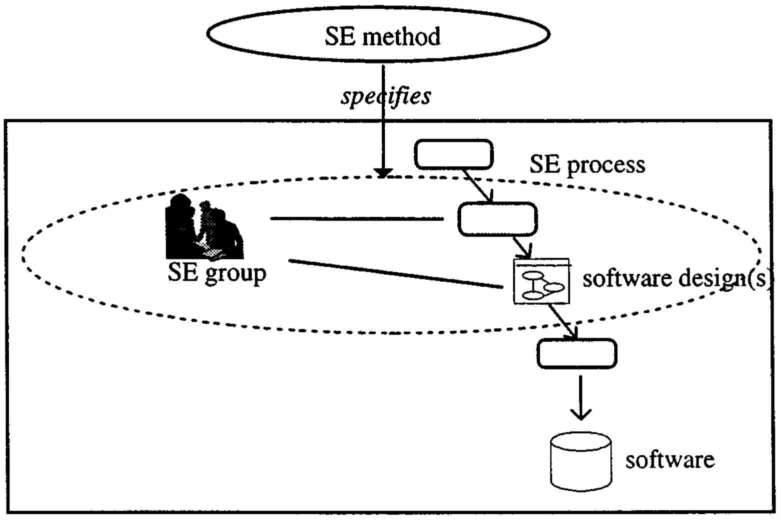


Figure 1. Software Engineering Methods in Relation to Software Engineering

At present, the use of software engineering methods is usually supported by computer aided software engineering (CASE[1]) technology. Basic CASE tools usually offer textual and/or graphical modelling capabilities employing one or a few methods. Thereby they help automate parts of the software development process. This is to increase productivity of software development and the quality of the products (see, for example, Marttiin, 1994; King and Galliers, 1994, p. 587; Kemerer, 1992, p. 23). Some have

[1] Forte and Norman (1992) define CASE as tools and methods to support an engineering approach to software development.

even believed that CASE tools will completely replace the software developer (see Vessey and Sravanapudi, 1995, p. 83). While this appears unlikely to us, we certainly deem CASE as a promising attempt to gain discipline for software engineering.

In multiple occasions, returns on CASE investments have fallen short of expectations. Orlikowski (1993) analyses these experiences from an organisational point of view. She sees the neglect to assess the development context as one reason for the failure. Marttiin (1994) confirms the lack of clear evidence to support claims about CASE productivity, and the tools' quality support is also deficient. He points out that the tools support too fixed a variety of techniques.

Despite this inflexibility, there are also more promising observations of CASE utilisation than those made by Orlikowski (1993). Iivari's (1996) experience of 35 Finnish organisations, however, shows that "[...] high CASE usage tends to increase the productivity and quality of IS and software development [...]" (Iivari, 1996, p. 101).

To conclude, methods are an effort to standardise the software engineering process (Kumar and Welke, 1992, p. 259; Harmsen, 1997). They concentrate mainly on the software products and may contain a process description of their application. Traditionally, they have focused on parts of the software life-cycle, but lately more comprehensive methods have surfaced. Methods are normally supported by CASE tools. Much of the CASE experience, however, has fallen short of expectations. One reason for this is the tools' inapplicability to different development situations such as projects of differing size, various programming environments and product software. This applies for the CASE tools supporting the methods by definition. Thus, method engineering is used to wield the methods.

2.4 Method Engineering

Method engineering is defined by Kumar and Welke (1992, p. 257) as "a method[ology] for designing and implementing information systems development method[ologie]s (ISDMs) i.e. meta-method[ology]. It is based upon the observation that organisations are continually devising, using, and adapting development method[ologie]s in their information systems development endeavours." Concordantly, Brinkkemper et al. (1995) and Harmsen (1997) speak of situational methods meaning project-specific methods and tools based on the observation that there exists no one right method for every development situation (Harmsen, 1997, p. 6; Kumar and Welke, 1992, p. 259).

Further, Harmsen (1997) distincts between method engineering and situational method engineering. He sees situational method engineering as a sub-area of the more general method engineering domain. Situational method engineering aims at controlled way to construct situational methods to certain projects while method engineering is not always directed to any specific situation.

Method engineering is moulding a method. This requires *metamodelling*, i.e., conceptualising of a modelling technique (Brinkkemper, 1990, p. 29). Metamodelling results in a *metamodel*, i.e. a conceptual model of the technique; a method specification. Marttiin (1994) divides this specification into three parts: a process model[1] describing the use of the method, a meta-datamodel covering the conceptual structures (language) used in the method and an agent model illustrating the role of the software engineers in the use of the method (organisation). Harmsen (1997) identifies these as well: the first two are denoted as method fragments among others and the last one, the human aspect, is referred to as one of additional concepts dealing with methods.

The following picture illustrates the relationship between software development and method engineering.

[1] Marttiin refers to activity model

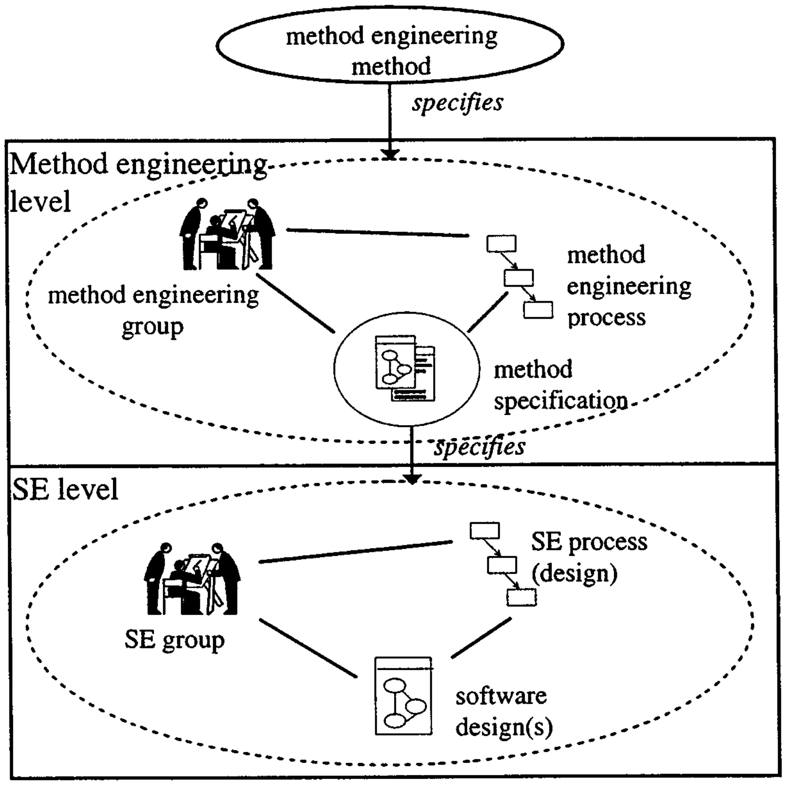


Figure 2. SE Method Engineering Architecture

Figure 2 presents the architecture for software design methods and method engineering (see Marttiin, 1994; Harmsen, 1997, p. 40). It has two abstraction levels. The lower one, the software engineering (SE) level, comprises the software engineering (SE) group, the design activities of a software process they execute and the software design(s) they produce. On the upper level, again, method engineering takes place. The method engineering group executes the method engineering process to produce a method specification to be used on the lower level. The method specification specifies some conceptual constructs and their use, i.e., it specifies the lower level software design process. As the higher abstraction level replicates the lower level pattern of process-group-product, it too can be method driven.

This is illustrated by Harmsen (1997, pp. 46-49), who describes a variety of approaches for constructing new methods or tools. These include the decision-oriented NATURE approach toward constructing method process models, ConceptBase and DAIDA for capturing and storing general methodological knowledge, PCTE aiming to provide a common interface and repository standards for CASE tools, Method Base for storing complete methods for situation dependent method selection and, finally, Task Packages and MERET for storing parts of methods for situation dependent method customisation and construction.

Method engineering and the technology to support it are subjects of growing interest. Computer Aided Method Engineering (CAME) (Marttiin et al., 1996) has been devised to support the method engineering task and metaCASE provides tools for both developing and using methods. (see e.g. Kumar and Welke, 1992; Marttiin et al., 1993; Marttiin et al., 1995; Rossi, 1995; Brinkkemper et al., 1995; Marttiin et al., 1996; Harmsen, 1997)

In conclusion, method engineering endeavours to accommodate methods to the organisation and situation specific needs. This appears to contradict with the standardisation aimed at by the methods. Therefore, when methods are moulded, it has to be taken into account that the modifying has to be in balance with the standardisation goal. All in all, method engineering involves modelling the process, concepts, and agents related to the method. As was the case with software engineering, method engineering is also supported by a variety of tools and methods.

In parallel with the methods and their development, a more abstract, process oriented research stream has evolved. The first signs of its emergence were life-cycle models, which are examined in the next section.

2.5 Life-Cycle Models

Definition. The first step in gaining control over the software development was to codify the "rules of thumb" used in the design activity so that they could be passed to future projects. This instructing was, however, insufficient to aid in the management of the process. Therefore, *life-cycle models* were developed. They describe the idealised structure of software development activities (Marttiin, 1994) by dividing the software process into distinct stages, such as specification and design, which allow the process to be better managed (see Hirschheim and Klein, 1992, p. 298). These phases are frequently supported by software engineering methods as we know them today.

History. Historically, Benington's stage-wise model (Benington, 1956) was probably the first creation to be called a life-cycle model. The most famous ones are definitely Royce's waterfall and Boehm's spiral model (Royce, 1970; Boehm, 1981) as well as prototyping approach (Budde et al. 1992). The International Standardisation Organisation (ISO) has also presented ISO 9000-3 guideline for the application of ISO 9001 to the development, supply and maintenance of software (see Kehoe and Jarvis, 1995, p. 4), which describes a general life-cycle for software development.

Evaluation. Life-cycle models describe the software process informally in a general level of detail. Madhavji (1991) offers an evaluation of them finding that "among the benefits of these life-cycle models are that they help us to become aware of, and gain an increased understanding of, the software process; and to determine the order of global activities involved in the production of software." (pp. 234-235). A limitation of life-cycle models is, as Madhavji (1991) puts it, that "they hide important process details that are crucial for the success of software projects." Consequently, life-cycle models are used more as a project management tool for project planning and control than to guide the actual software development. According to Curtis et al. (1992, p. 75), they are insufficient for this purpose as well, owing to their concentration on product engineering.

Figure 3 illustrates the relation of life-cycle models to the software engineering (SE) process, which produces software and is executed by software development group. The model specifies how the process should be executed. Traditionally, life-cycle models have excluded the agents of the process and the more detailed description of the products, representing an abstracted view of the process itself.

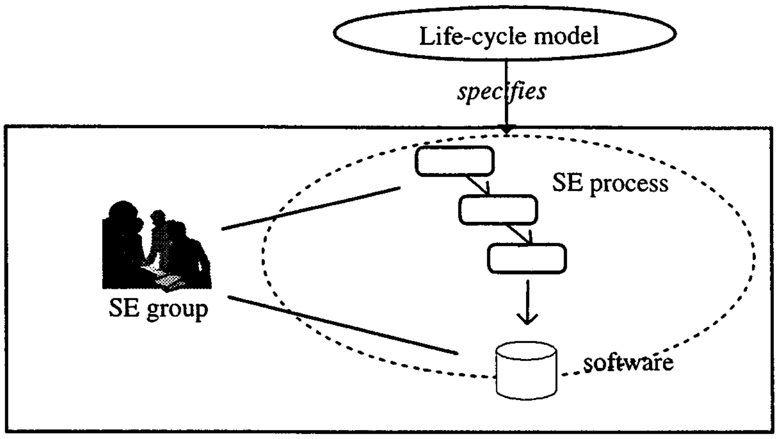


Figure 3. Life-Cycle Model Driven Software Process

In this study, life-cycle models are considered as a pre-historic phase in the history of software process modelling. This is not to imply that they would not still be in use and serviceable today, because they are. However, they represent only a fraction of the world of software process modelling, the most traditional part of it. Therefore, life-cycle models are not given more attention in our report than this simple presentation here.

The birth of software process modelling as we know it today was the accomplishment of the software process movement, which is the next subject in our examination.

2.6 Software Process Movement

"A *software process movement* [italics added] emerged in the mid-1980's when the shortcomings in managing the development process were recognised as a prime inhibitors of growth in software productivity and quality." (Curtis et al., 1992, p. 85; Curtis, 1992, p. 335; also Curtis and Paulk, 1993, p. 381)

Process quality vs. product quality There are two basic approaches to develop the quality of software. These are to improve the product quality and to improve process quality. At present, there is a wide consensus among researchers as well as practitioners involved in software engineering, that the quality of a software product to a high degree relies on the quality of the software process which produces it. This is demonstrated by the multitude of workshops and conferences dealing with the issue of software process. To mention some, there have been four International Conferences on the Software

Process (Proceedings ICSP, 1996), ten International Software Process Workshops (Proceedings of ISPW, 1997) as well as five European Workshops on Software Process Technology (Proceedings of EWSPT, 1996), and a European Workshop on Software Process Modelling (Proceedings of EWSPM, 1991) (see e.g. Heineman et al., 1994; Humphrey, 1989; Madhavji et al., 1994; Kellner and Hansen, 1989; Höltje et al., 1994; Ellmer, 1995; Conradi et al., 1993; Curtis and Paulk, 1993; Haikala and Märijärvi, 1995). Hence, many of the current research efforts are aimed at improving the quality of the software process.

Critique of life-cycle models. As a result of the shift of focus to the process, life-cycle descriptions were discovered not to fulfil the needs of the software engineering community. Most of them present an extremely abstract model of software development and do not provide clear guidance on how to integrate the many process steps that project staff performs in software development (Curtis et al., 1992, p. 76). Moreover, life-cycle models are typically linear and thereby inhibit the representation of feedback loops and iterations which are typical in actual software processes. Furthermore, as Krasner et al. (1992, p. 98) remark, these models tend to focus on the series of artefacts that exist at the end of the process, rather than on the processes that are conducted to create the artefacts.

Madhavji (1991, p. 235) lists a number of details left uncovered by life-cycle models. These include:

- the process steps required to resolve customer reported software problems;
- triggering and terminating conditions of an activity;
- the state of a product component before, during, and after a process step;
- the notational, methodological, operational and other tools to be used in different process steps;
- the inputs and outputs of an activity, and the sources and destinations of the data;
- the manner in which data flow from one activity to another;
- the roles played by humans in the process;
- the constraints on the process steps;
- how communication among humans is supported;

- where parallel and sequential process steps exists.

For more critique on life-cycle models see, for example (Humphrey, 1989, p. 249; Curtis et al., 1992, pp. 75-76; Tamai, 1993, pp. 339-340; Curtis et al., 1987, p. 96; Deiters and Gruhn, 1994, p. 230)

In the following, our focus moves to the assessment frameworks and improvement approaches, which can be characterised as the most comprehensive contributions of the software process movement.

2.7 Software Process Assessment Frameworks and Improvement Approaches

The shift of focus to the software process has lead to the development of various software process assessment frameworks and improvement approaches. These provide the means for evaluating the software process and for elevating the process quality. They aim at continuous process assessment and improvement.

Assessment frameworks. The Capability Maturity Model (Paulk et al., 1993; Paulk et al., 1995), SPICE (Dorling, 1993) and the Bootstrap approach (Koch, 1993) are at present perhaps the most significant assessment frameworks. Their aim is mainly in evaluating software processes and setting the direction for improvements.

CMM. In literature, CMM is the most used and cited software process assessment framework (see, e.g., Lai, 1993; Herbsleb and Goldenson, 1996; Kitson and Masters, 1993; Ellmer, 1995; Curtis, 1992; Jorgensen, 1990). The CMM is based on the maturity framework which was first elaborated by Philip Crosby in his famous book "Quality is free" (1979)[1]. The CMM describes five levels of software process maturity. Each level is a well-defined evolutionary plateau on the path toward mature software organisation.

Improvement approaches. In addition to assessment frameworks, there exists several improvement approaches. The most popular one is perhaps the Quality Improvement

[1] For more about the CMM's history and its evolution see for example Curtis and Paulk (1993, p. 382).

Paradigm (Basili and Rombach, 1987; Basili and Rombach, 1988). A fairly similar "improvement process" is given in (Humphrey 1988; 1989). These two define a sequence of steps to improve the software process (Kontio, 1995, pp. 29-30; Lai, 1993, p. 16; Bandinelli et al., 1995, p. 440; Heineman et al., 1994, p. 512; Madhavji et al., 1994). The steps of Humphrey's "improvement process" are as follows:

1. Understand the current status of the development process or processes.
2. Develop a vision of the desired process.
3. Establish a list of required process improvement actions in order of priority.
4. Produce a plan to accomplish the required actions.
5. Commit the resources to execute the plan.
6. Start over at step 1.

In addition to these perhaps the most famous ones, numerous other approaches have been devised. Here are a few examples. First, there is the Quality Improvement Paradigm based Experience Factory (Basili et al., 1994), which focuses on software process improvement founded on learning from the process experience. Second, Humphrey (1995) has developed a Personal Software Process for improving an individual's software process. Third, McFeeley (1996) formulates a continuous loop through the steps necessary for software process improvement as a model named IDEAL. Fourth, the famous Plan-Do-Check-Act paradigm uses feedback mechanisms to improve processes and, finally, the equally far-famed Total Quality Management (TQM) represents a management approach for improving process quality which involves all members of an organisation (see Pohl, 1996, p. 52). For more about assessment frameworks and improvement approaches see (Kontio, 1995; Pohl, 1996).

2.8 Summary

In this chapter, we have presented the theoretical background of software process modelling. We have included those issues conceived to guide the researchers and practitioners of the software engineering field in their efforts to develop software process modelling to its current state. The main efforts striving to improve the quality of software include methods and CASE tools supporting them, life-cycle models as well as software process assessment frameworks and improvement approaches. Now, the historical review has arrived to the point where software process modelling steps in.

In thirty years, the software engineering domain has advanced a long way from a craft-like ad hoc software development towards a disciplined approach. No silver bullet to cure the software crisis has nevertheless been found. More recently, the research has turned away from attempting to find one and it has been accepted that multiple models, methods and tools are needed. Furthermore, there has been a change of focus: instead of software product quality, most efforts now concentrate on software process quality.

> After two decades of unfulfilled promises about productivity and quality gains from applying new software methods and technologies, software development organisations are realising that their fundamental problem is their inability to manage the software process (Curtis, 1992., p. 335)

This software process movement was the incentive for the introduction of several assessment frameworks and improvement approaches to aid in improving the process quality. They provide abstracted tools for managing the assessment and improvement of the software process.

The next stage in our examination is the field of software process modelling, which is resultant from the process movement. The shortcomings of the means presented in this chapter to properly facilitate the management of the software process have influenced the transition of research emphasis to this young field.

3 SOFTWARE PROCESS MODELLING

"*Software process modelling* [italics added] [...] explicitly focuses on phenomena that occur during software creation and evolution, a domain different from that usually modelled in human-machine integration of information systems design" (Curtis et al., 1992, p. 75). It produces a *process model*, which "is an abstract description of an actual or proposed process that represents selected process elements that are considered important to the purpose of the model and can be enacted i.e. executed by a human or machine" (Curtis et al., 1992, p. 76).

Software process modelling research has lived its heyday for more than ten years. Several software quality workshops and conferences (see Section 2.2) have dealt with the issue and even one workshop dedicated specifically to modelling has been held (Proceedings of EWSPM, 1991). The first major discussion concerned with software process modelling occurred in the International Conference on Software Engineering in 1987 when Leon Osterweil gave his famous presentation "Software processes are software too" (Osterweil, 1987). He also expressed that there is no "ideal software process description" to suit every development situation (p. 10), which has been an established truth in the modelling domain. Ever since this major discussion, software process modelling has been a popular theme. This applies not only for the software engineering research community but also for the software industry (King and Galliers, 1994; Bandinelli et al., 1995; Christie, 1993; Kellner and Hansen, 1989).

Relation to past research. The following paragraphs outline a plausible causal connection between software process modelling and the innovations presented in the previous chapter. Software process modelling, however, is a multidimensional phenomenon, and an analysis such as the one below reveals only a fraction of the implications software process modelling has had on the software engineering theory and practice. Its significance will be examined more closely later in this chapter (Section 3.3).

Complexity gap. From the perspective of life-cycle models and methods, Raccoon (1995a, 1995b) sees software process modelling as filling the complexity gap between these two descriptions of the software process depicting it on different levels of abstraction or complexity (Raccoon, 1995a, 1995b). On the one hand, modelling is used to decompose the traditional large-grained life-cycle descriptions into sufficient level of detail so that they can provide more explicit guidance for executing software development projects. On the other hand, modelling is to guide the use of methods and CASE tools as well as their accommodation into the software process. The same idea has obviously inspired the CMM's definition of the software process, which the model is supposed to depict. According to the CMM, a software process is a set of activities, methods, practices and transformations to develop and maintain software and the associated products (e.g., project plans, design documents, code, test cases, and user manuals) (Paulk et al., 1995, p. 365).

As mentioned in Section 2.3, recent development has, however, brought methods closer to software process models. Harmsen (1997, p. 5) reports that "in the mid-eighties, methods covering larger parts of the systems development process appeared, including project management." Ideally, the resumption of such development would lead to all-embracing methods covering the entire software development process. Then, the process model included in the method would represent all of the software process and the complexity gap would vanish. Still, as the software development situations vary, a ready-made comprehensive method remains a practical impossibility. Thus, there will always be the need for detailed software development methods to provide more technical guidance to the development and more abstract models to give an overall picture of the software process. They render it easier to fit the various methods and techniques into the process. This is what Raccoon denoted as the filling-the-complexity-gap-function of process modelling.

Assessment and improvement. The relation of software process assessment frameworks and improvement approaches to modelling is more evident than that of software engineering methods. Generally, they see software process modelling as a means to evolve the process. Improvement approaches more explicitly use modelling to characterise the current status of a software process.

Next issues. The subsequent two sections place the software process modelling phenomena in their wider context and make explicit their relationship to the software engineering domain. The first one utilises a software process modelling architecture to relate modelling to the software engineering domain. The second one clarifies the associations of software process modelling in the software process engineering domain.

Then, the following sections will first discuss the possible objectives of software process modelling (Section 3.3) and the emphasis of previous research (Section 3.4). After that, perspectives embedded in process modelling languages are scrutinised (Section 3.5). Subsequently, the languages themselves are examined and means to select one among them are offered (Section 3.6). The two approaches to process modelling, software process modelling and software engineering method engineering, are discussed and compared (Section 3.7). Finally, in Section 3.8, a summary of the process modelling research field is provided, which emphasises the need for method driven software process modelling.

3.1 Software Process Modelling Architecture

According to Section 2.4, a method specification contained a process model specifying a part of the software process. The creation of a metamodel (i.e. a method specification) took place on a higher level of abstraction than the use of the method. The same differentiation of abstraction levels was repeated in Section 2.5. There, life-cycle descriptions were deemed to specify the software process and were thus pictured as being above the ordinary software process level in Figure 3. Similar meta-level thinking can be found already in Auramäki et al. (1988).

The architecture in Figure 4 below is analogous to the situations referred to above. It illustrates the relationship between software engineering and software process modelling. The domains are rather similar to each other, the difference mainly being a shift in meta level (Conradi, 1994b).

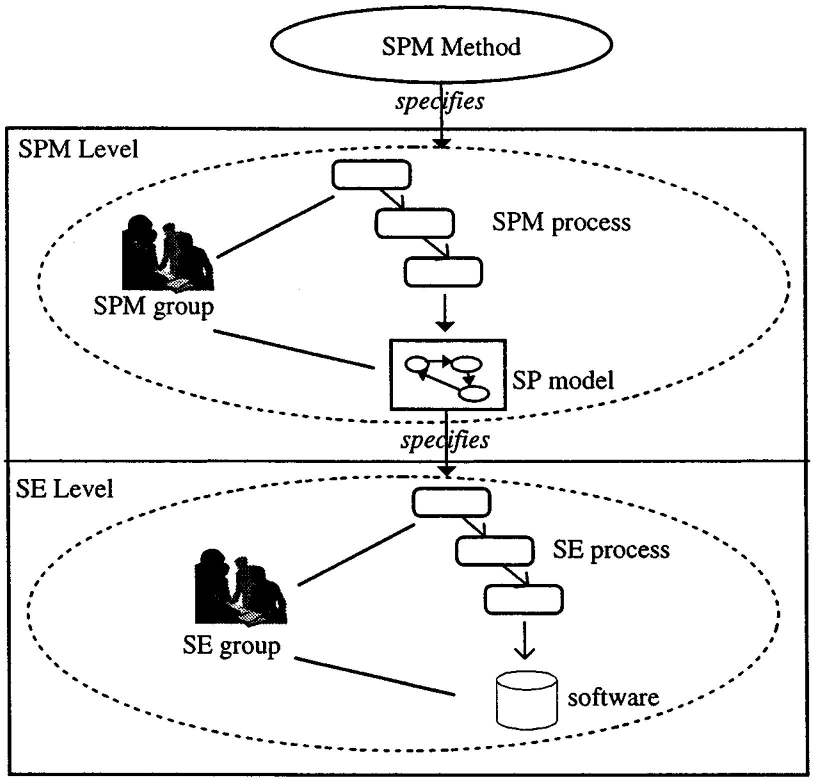


Figure 4. Software Process Modelling Architecture

Figure 4 is divided into two abstraction levels. First, the *Software Engineering (SE) Level* is where the actual software process, all the real world activities i.e. elementary process steps involved in software development and maintenance, are enacted by the software process actors. They constitute the *Software engineering group*. Here are a few terms to be clarified. Firstly, enactment means the execution or interpretation of a software process model (Lonchamp, 1993). Secondly, the term actor or agent, denotes a human or a machine who enacts a process (Curtis et al., 1992, p. 76). Role, again, is a responsibility assigned to a process actor. Thus, a single process actor can have multiple roles.

Second, at the *Software Process Modelling (SPM) Level*[1] (also termed meta-process level, see Section 3.2 Software Process Engineering) a *software process modelling group* executes a *Software process modelling process*, and produces a *software process (SP) model*. This model is constructed using a process modelling language (PML) (see Section 3.6) by following the description of its use, which is usually incorporated in the

[1] In her framework, Koskinen (1996, p. 16) refers to this level as ISD process modelling (level).

PML, at least implicitly. Usually, this process description begins with something like identifying activities and grouping them and goes on to identify activity inputs until the process model diagram has been constructed.

Modelling classification. Basically, depending on the nature of the resulting model, all software process modelling falls into one of three categories. Modelling may employ a descriptive, a prescriptive or a proscriptive view of the process. In modelling practice, these categories and views are generally combined in appropriate proportions. The choice of the main viewpoint is based on the objectives of the modelling (discussed in Section 3.3).

First, d*escriptive modelling* attempts to model the process as it is. Second, *prescriptive modelling* strives to describe how the process should be (King and Galliers, 1994, p. 587; Lonchamp, 1993). Finally, *proscriptive modelling* models what is not allowed in process. This last class is a special case of the second one: modelling prohibitions it negates the prescription (Lonchamp, 1993).

Traditionally, research emphasis has focused on prescriptive modelling as is illustrated, for example, in Section 3.4 Automated Guidance and Execution Support – Process Programming. Recently, however, descriptive modelling has received increasing attention. This issue is touched upon in sections 3.3 Software Process Modelling Objectives and 3.5 Software Process Modelling Perspectives.

Third level of abstraction and meta-circularity. In addition to the two abstraction levels presented above, there may be more. This matter is elucidated in Figure 4 with the *software process modelling (SPM) method* above the software process modelling level, meaning that the software process modelling process itself is method driven. Thus, as Kellner (1996, pp. 38-39) points out, these abstraction levels of software engineering are actually repetitions of the same pattern on different meta-levels. In principle, one could go on endlessly adding new meta-levels replicating the pattern of group-process-product. To avoid meta-circularity, however, we halt to two levels for now (see Lonchamp, 1993).

The following section delves into the second meta-level, the software process modelling level, in the wider context of software process engineering.

3.2 Software Process Engineering

Software process engineering is a discipline of software engineering dedicated to the study and improvement of software processes (Lonchamp, 1993). The definition of software process engineering is, however, still evolving. Thus, to stress the continuous development of software processes, Conradi et al. (1993) speak of process evolution to denote software process engineering. Their evolution means customising the process to accommodate requirements or preferences or to cope with various unanticipated situations, which is a goal of software process engineering. For more definitions of the area see, for example, Madhavji (1991, p.235), Starke (1993, p.13), Heineman et al. (1994, p. 503) and Kellner (1996, p. 39).

Engineering, improvement and modelling. According to Lonchamp (1993), software process engineering aims at improving the software process. Process improvement has already been mentioned above, associated with process improvement approaches (in Section 2.7). Ellmer (1995, p. 82) identifies process modelling, process evolution and process monitoring[1] as the most important functions for process improvement. Thus, software process engineering provides a practical framework for the use of process modelling in process improvement.

Software process modelling is a significant and integral part of software process engineering. Improvement approaches, such as Quality Improvement Paradigm and Humphrey's improvement process, use modelling as a tool for understanding the current status of the process and to prescribe how the process should work. The notable role of modelling is confirmed by Kellner et al. (1996, p. 37) as they remark that "a crucial focus of SPI [software process improvement] efforts is on the design, definition, and evolution of software processes." All of these goals engage software process modelling.

[1] Monitoring denotes collecting quantitative data of the execution of a software process.

Now, having defined software process engineering as a discipline and the role of software process modelling in it, let us turn into examining the relationship of software process engineering domain to that of software engineering.

Software process engineering in relation to software engineering. To illustrate the role of software process engineering in relation to software engineering Madhavji (1991) exhibits his process cycle (also denoted as process life-cycle), which defines the scope of the total set of process steps necessary for the development and evolution of software processes (Madhavji, 1991, p. 238-240). The process cycle is shown in Figure 5 below.

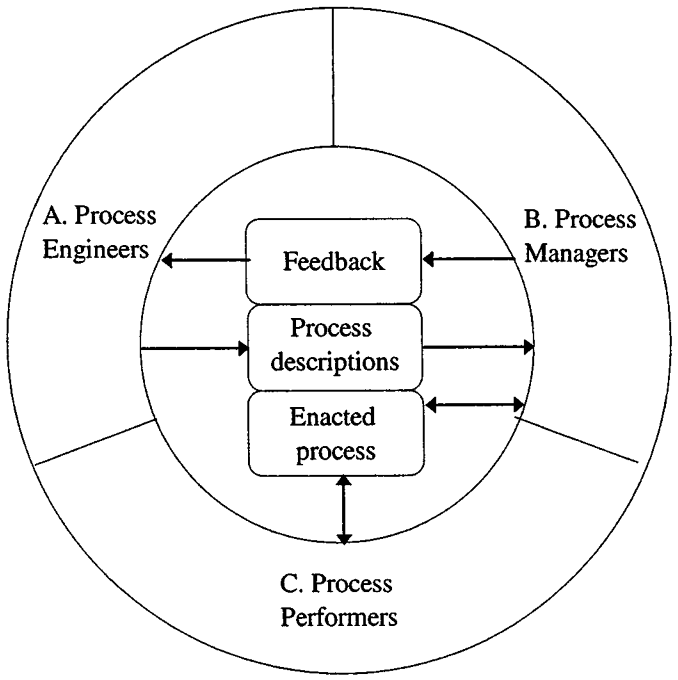


Figure 5. The Process Cycle ((Madhavji, 1991, p. 240)

Madhavji's process cycle extends the traditional view of software engineering to include the software process engineering aspects. It consists of three sectors, A, B and C, representing the process engineering, process managing, and process performing activities of software engineering, respectively.

"In sector A, process engineers design, construct, and improve generic process models (or models when the context is clear). These models are generic in the sense that they have not yet been tailored for a particular software project. The models are customised by process

> managers in sector B and are instantiated for use; process performers in sector C carry out the processes." (Heineman et al., 1994, p. 503).

The process cycle also pays attention to feedback generated by process performers, which is used to improve processes (Madhavji, 1991, p. 239-240). In Figure 5, this is shown with arrows from performers and managers via "Feedback" in the middle to process engineers.

Meta-process. Kellner et al. (1996, p. 38) give another illustration of the relationship between software process engineering and the software process. They speak of the software process engineering process, which is termed meta-process[1]. This meta-process is where software process engineering activities take place. Furthermore, Kellner et al. (1996) suggest a method for software process engineering[2]. The authors also argue that the software engineering community is only beginning to realise the importance of software process engineering to the software discipline. Hence, it appears likely that more such approaches will emerge in the near future. The Quality Improvement Paradigm (see Section 2.7) offers a practical example of a meta-process. It may be used by process engineers to model and develop the software engineering process.

Next issue. This section and the one above have placed the software process modelling phenomena in their wider context and made explicit their relationship to the software engineering domain. The next issue in our course of examination, the objectives of software process modelling, discusses more specifically the purposes modelling can be used for within that context.

---

[1] The same term is used by Conradi et al. (1993, p. 28) to denote a process whose goal is to maintain and evolve the whole software process, i.e. the software production process, the software process support and the software meta-process itself (see also Nguyen and Conradi, 1994). Their software meta-process is thereby considerably more comprehensive than the definition we have accepted. For Lonchamp (1993), on the other hand, meta-process means a process describing the engineering of software processes as well.

[2] In their terminology, the method is for software process improvement engineering, which excludes some software process engineering activities such as identifying, generalising and documenting software process components for potential later reuse (Kellner et al. 1996, p.39).

3.3 Software Process Modelling Objectives

There are a multitude of objectives software process modelling can be perceived to serve. The categorisation of modelling objectives Curtis et al. (1992) present is perhaps the most popular in process modelling literature (see, for example, Heineman et al, 1994, p. 502; Kontio, 1995, p. 33; Krasner et al., 1992, p. 91; Koskinen, 1996, pp. 22-24; Marttiin, 1994; Yu and Mylopoulos, 1994, p.159). The authors introduce a five category list of objectives and specify the list with 25 goals. Their five item list is not exhaustive, but gives a comprehensive overview of the uses of software process modelling[1]. The categories are:

1. *Facilitate human understanding and communication.* A depiction of a process may describe any facet(s) of it. This abstraction and the presentation of connections between real-world entities render the process more readily understandable, and thereby also facilitate communication of the process. All the software process improvement approaches presented above in Section 2.7 first aim at understanding the process.
2. *Support process improvement.* The process model may be used to develop the process itself. The improvement targets can be identified and potential improvements analysed using the model. Maintaining a comprehensive picture of the process facilitates process component reuse, process comparison, technology insertion and managed process evolution.
3. *Support process management.* A depiction of the process helps managers to plan, control, monitor and co-ordinate the process. In fact, Lott and Rombach (1993, p. 407) maintain that the lack of explicit software process models is one major reason why software development and maintenance projects are difficult to plan and manage[2].

[1] In his Licentiate's Thesis, Kontio formulates a more exhaustive list of objectives, which he divides into ten classes. Other classifications are given in Armenise et al. (1992) and Kellner and Hansen (1989), for example.

[2] Other major reasons include the lack of operational definitions of target qualities and the lack of tractable quality models.

4. *Automate process guidance.* Formal process models provide the means to automatically guide the process actors by suggesting actions and reference material or invoking tools, for example.
5. *Automate execution support.* Furthermore, from a process model it is possible to identify process parts to be automated. Similarly to automatic guidance, formal models render it possible to computerise the execution of some process activities. (Both automatic guidance and automatic execution support are discussed in more detail under the title of Automated Guidance and Execution Support – Process Programming in Section 3.4).

Selecting objectives. All these objectives ultimately strive to improve the software process so as to improve the quality of its end deliverables. If modelling is to serve its purpose, whatever it may be, some of these interrelated objectives should be emphasised more than others. This focusing should be based on the requirements, which modelling and the resultant model are supposed to meet in a specific subject environment.

Curtis et al. (1992) do not offer any priority order for their objectives, except implicitly by stating "Unfortunately, interest in facilitating human understanding and communication has received less attention from the research community than has machine enaction. Process models and definitions can not be used if they cannot be understood." (Curtis et al., 1992, p. 83). From this it can be concluded that understanding is the foundation that all the other objectives build on. The importance of the goal is promoted more explicitly by Heineman et al.:

> "Even simple processes quickly become complicated because there can be many processes performed simultaneously by different people, so that often no one person understands the entire set of processes. Consequently, gaining understanding is a valuable goal in its own right." (Heineman et al., 1994, p. 506)

Similar observations have been made, for example, by Kellner et al. (1996) and Kontio (1995). The significance of understanding is expressed also in Kellner and Hansen (1989), Madhavji (1991) and Höltje et al. (1994).

The selected modelling objectives affect the choice of the modelling language, which is the subject of Section 3.5. The next section, however, more closely discusses the last two objectives in the list by Curtis et al. (1992): automate guidance and execution support. As implied above, they have been favourites in previous research.

3.4 Automated Guidance and Execution Support – Process Programming

Process programming refers to creating explicit and formal software process models, which can be interpreted to provide automated guidance to the software developers (see Lonchamp, 1993[1]) and automate parts of the software process (see Heineman et al., 1994).

This paradigm was presented by Osterweil (1987) and is currently the focus of an entire community of researchers (see, for example, Kontio, 1995, p. 14-15; Heineman et al., 1994, p. 517 and Emmerich et al. 1996, p. 2 for observation on the popularity). The flourishing research has produced a number of process-centred software engineering environments[2] (PSEEs), most of which comprise a representation formalism (see Armenise et al., 1993). They also offer a means to build and analyse software process models to be enacted in the computerised environment.

> An important aspect of such a process-oriented environment is that all the elements of a software project are explicitly integrated through the medium of software process: for example, human beings in their various technical and managerial roles; software tools, product parts and their states at various times in the process; timing, budgetary, data source and destination, goals, policies, and other constraints acting on the development process; the development activities and the order in which they are to be carried out etc. (Madhavji, 1991, p. 236)

Examples of PSEEs include IStar supporting a contractual approach (Dowson, 1987), SPADE embedding the Petri-net based SLANG modelling language, which takes into

---

[1] Lonchamp (1993) speaks of process-centred software engineering environments, which are an instance of the process programming paradigm.

[2] Also referred as process-centred support environment (PSE) or process-centred environment (PCE). Conradi et al. (1993) use the term process support technology, which constitutes software process support with the process model. The technology is used to define, modify, analyse, and execute the model.

account process evolution (Bandinelli et al, 1993; Emmerich et al., 1996), and MERLIN using rule-based technique to model and enact a software process (Junkermann et al., 1994). The hybrid environment EPOS, again, combines features found in many current PSEEs (Conradi et al., 1994a). Further, Kaiser et al. (1996), introduce a metalinguistic approach allowing the engineering of the modelling language built in a PSEE. For more about PSEEs and evaluations of them see Nguyen and Conradi, (1994), Lott (1993) and Armenise et al. (1993). Also Finkelstein et al. (1994) collect European state of the art projects and systems addressing modelling and technology issues. Their book includes system assessments as well (Lonchamp, 1994b). The systems examined comprise EPOS, SOCCA, MERLIN, OIKOS, ALF, ADELE-TEMPO, SPADE, PEACE, E3 and PADM.

Perhaps the most significant accomplishment of the process programming paradigm is that they have proved that parts of the software process can be automated (Heineman et al., 1994, p. 520). As was indicated in the previous section, however, PSEEs are only capable of serving a subset of process modelling objectives. Nevertheless, process programming and PSEEs are and will be a popular field of research. According to Pohl (1996, p. 61), the need to technically support process improvement by suitable software engineering environments is more and more recognised.

Most PSEEs embed one or more process modelling languages, which usually are hard coded in them. Many of them also utilise a coarse modelling method. These characterise an approach to process modelling. Before discussing the languages in detail, however, we examine the different perspectives process models and, thus, process modelling languages, embed.

3.5 Software Process Modelling Perspectives

Modelling perspectives offer different points of view to the same software process. Each perspective exhibits a specific set of the process phenomena. To capture all the software process information perceived relevant, multiple perspectives are useful (Heineman et al. 1994, p. 509). In fact, the models usually combine two or more perspectives to meet their requirements. The process details depicted in a model and, hence, the perspective(s), largely depend on the modelling language used. Similarly, we could thus speak about software process modelling language perspectives.

Curtis et al. (1992) list four most commonly used perspectives into the software process[1]:

Functional perspective concentrates on *what* happens in the software process. This is a static view of the process describing what activities are performed and what flows of informational entities (such as artefacts, documents) are relevant to these activities. Functional perspective is traditionally represented with Data Flow Diagrams (DFDs).

Behavioural perspective illustrates *when* the activities are performed and *how*. The behavioural view of the process is usually described with state transition diagrams.

Organisational perspective concentrates on *where* and *by whom* in the organisation activities are performed. This perspective describes the structural view of the process and utilises module charts, for example.

Informational perspective represents the informational entities produced or manipulated by the activities.

Other perspectives. Kontio (1995) proposes a scheme differing from the one above. He suggests modifications to the notion of using process perspectives based on process information comprising process activities, process behaviour, process artefacts, process agents, process resources, process infrastructure and process information flow.

> The definition of perspectives should be based on the process model information entities I have defined. Theoretically, there are 127 possible perspectives as the sum of binomial coefficients shows:

---

[1] The list is cited also in Heineman et al. (1994, p. 509) and Koskinen (1996, p.28). Krasner et al. (1992) present a similar construct as well. They define a plan including data on what is to be built, who is available to do the work, where, when and how the work is to be performed as well as why.

number of perspectives = $\sum_{j=1}^{7} \binom{7}{j} = 127$

However, not all combinations of process model information entities are meaningful, so the number of useful process perspectives is significantly smaller.[...] the most relevant ones, i.e., the number of possible, relevant perspectives seems to be somewhere between 20 and 30. (Kontio, 1995, p. 66).

Completeness. The choice of modelling perspectives, i.e., which process information to model, is strongly connected to the choice of modelling objectives. Modelling perspectives, again, affect the completeness of the model. "Completeness depends on whether all important components of information remain after less crucial details have been abstracted away by the modelling technique, and on whether the intended uses of the model are enabled by the perspectives offered." (Curtis et al., 1992, p. 77). This calls for a multi-perspective approach. Actually, when j >= 2 in Kontio's formula above, his perspectives also represent multiple viewpoints to a software process.

Although combining several perspectives results in a more complete model, the representation may also be confusing in its richness: too much information on one diagram inflicts overload and none of the information is properly conveyed (see Koskinen 1996, p. 28). To prevent this "diagram-clutter", Curtis et al. (1992, p. 77) submit that the perspectives should be modelled separately yet interrelated. Controlling the simultaneous use of a number of perspectives is perhaps better facilitated when the perspectives are clearly bound to the process information (entities) presented, as in Kontio's (1995) scheme above.

As noted above, the perspective(s) of a software process model largely depend on the modelling language used. The perspectives relate to process concepts, which are represented with modelling languages. This leads us to the next section.

3.6 Software Process Modelling Languages

Armenise et al. (1993) provide a compact characterisation of software process modelling languages.

> A successful modelling requires the use of an adequate notation or language. Notation can be either textual or graphical or mixed text and graphics. Some notations have a well-defined syntax and thus qualify as languages. Some languages have formally defined semantics and therefore are amenable to formal reasoning. (Armenise et al., 1993, p. 402).

Numerous process modelling languages (PMLs)[1] have been developed to tackle different software process modelling objectives (see for example Bandinelli et al., 1993; Deiters and Gruhn, 1994; Christie, 1993; Shepard et al., 1992; Dutton, 1993; Kaiser et al., 1993; de Bunje et al., 1996). Beside process technology, languages have been the main focus of research (Pohl, 1996, p. 65; Kontio, 1995, p. 15). This owes to the non-existence of an ideal software process model which could be replicated to all software development situations and the programming-orientation of modelling research. Most of these languages are formal with well defined syntax and semantics and serve process enaction more than description. They borrow characteristics from several programming and modelling paradigms, such as conventional and real-time programming, system analysis and design, database modelling languages, artificial intelligence languages and approaches, Petri nets, precedence networks etc. (Armenise et al., 1993, p. 409; Armitage and Kellner, 1994, p. 153).

Software processes, however, have characteristics distinct from the application areas of these more traditional linguistic approaches. Therefore, in addition to features similar to these, the PMLs are usually extended by adding software process specific aspects such as support for both formal and informal descriptions, support for controlled modifications during enactment and representation of both technical and nontechnical activities (Armenise et al., 1993, p. 409).

The vast number of PMLs developed and the endlessly wide spectrum of different modelling situations renders the choice of a suitable language for modelling a real challenge. Some means are provided by the variety of classification attempts presented in literature. But, as Starke (1993) aptly expresses, there is no expedient to substantially facilitate the evaluation.

[1] The term language is often used as a synonym for notation and formalism. We use the terms interchangeably as well, although we agree to the distinction Armenise et al (1993) point out.

> A variety of about 60 different languages exist to model software processes. Several articles have tried to clear up that language-jungle, but still no approach exists to systematically evaluate the applicability of these languages to the real process modelling problems [goals]. (Starke, 1993, p.13)

In fact, a universal framework is practically impossible to devise. In principle, it should evaluate and match two sets of properties: those of the modelling situation including its needs and assets and those of the modelling languages. As especially the former, but also the latter are rather unwieldy for analysis and classification, no framework can be exhaustive, but has to make crucial presuppositions of the modelling situation and is likely to emphasise some aspects of modelling languages over others.

Nevertheless, these efforts may ameliorate the selection process by offering some decision criteria. The state of classification is reviewed in the next subsection.

3.6.1 Means to Select a Suitable PML – Classification Attempts

To provide an overview of modelling techniques, Curtis et al. (1992) review five approaches to representing process information.

1. *Process programming approach.* Strongly influenced by the process programming paradigm introduced by Osterweil (1987), the process programming community argues that software processes should be programmed and executed as computer programs. APPL/A, for example, is an ADA based process programming language developed to construct process programs (see also Armenise et al., 1993, pp. 411-412).
2. *Functional approach.* The Hierarchical and Functional Software Process (HFSP), for instance, provides a functional approach to software process modelling (see Armenise et al., 1993). It describes the software process as a collection of (hierarchical) activities and information flows between them.
3. *Plan-based approach.* Emerging from the artificial intelligence research, this approach describes a process as a collection of plans, preconditions and constraints to achieve goals. GRAPPLE exemplifies a constraint-based approach to process modelling (see Curtis et al., 1992, p. 81).

4. *Petri-net approach.* This approach supports the description of roles and their interactions. See SLANG (Bandinelli et al., 1993) or FUNSOFT (Deiters and Gruhn, 1994), for example.
5. *System dynamic approach.* In this approach, feedback and control system techniques are applied to software process modelling. It also involves quantitative representations.

Hybrid approach merges two or more of the approaches above and thus integrates multiple representational paradigms. This, however, raises additional challenges regarding translation, communication, co-ordination, and interaction among model components. Examples of this approach are ProNet (Christie, 1993) and SOCCA (Engels and Groenewegen, 1994). Currently, the hybrid approach is considered necessary for effective software process modelling (Curtis et al., 1992, p. 85). Thus, it constitutes a sixth category to the list above.

Other classifications. As McChesney (1995) points out, the illustrative list above and many other classification efforts presented in the literature (see, for example, Madhavji, 1991; Armenise et al., 1993; and Avrilionis et al., 1996) categorise software process modelling approaches based primarily on the style of the PML. He finds eight distinct styles used as classification criteria: imperative programming, rule-based, AI/knowledge-based, Petri net-based, functional programming, object-oriented, formal specification, mathematical modelling. McChesney (1995) makes a clear distinction between the concepts PML and process modelling approach. His message is that such classifications should also take into account modelling objectives, process features that can be modelled and the worldview associated with the approach. The author also presents a scheme for such a classification. In our opinion, however, such a wide scale of modelling approach attributes are better addressed as a software process modelling method (see Chapter 4).

Yet another classification is proposed by Koskinen (1996, p. 30). Her framework differs from others, as it combines process modelling focus (processes, products or agents) and the role of the approach in relation to Galbraith's theory of organisational design.

To conclude, the wide spectrum of classification attempts has not been able to submit comprehensive and universally applicable categories for PMLs. As the next subsection will show, this stresses the importance of modelling objectives in the selection of a suitable PML.

3.6.2 Means to Select a Suitable PML – Desirable Approach Characteristics

Kellner and Hansen (1988; 1989) have established general requirements for software process modelling approaches based on software process modelling experience. Their list covers a wide spectrum of modelling approach characteristics, such as visuality, compendiousness, formality, analysability and simulatability. Approaches should support multiple paradigms and levels of abstraction, model configuration management and integration to others. Furthermore, the authors give some desirable representational capabilities.

The list above is intended to provide general requirements and desired characteristics for a PML. Some of them, however, are substantially less significant than others, depending on the modelling objectives. This is illustrated with two such issues, abstraction and formality, which are discussed in the following paragraphs. Both are closely related to the enaction of software process models and can thus be deemed as valuable features for the PML to possess.

Abstraction brings up the question about granularity and precision, in other words the size of the process elements represented in the model and the degree to which a process description specifies all the activities needed to produce the accurate result. For automated process enaction, a small-grained and precise model is usually necessary, while a process to be executed by human actors can be represented with a more large-grained model, because human actors are expected to possess enough process knowledge to enact the model. They are thus provided the possibility to tailor the model for their needs. (Curtis et al., 1992, p. 83).

Formality is a characteristic advantageous if the model is to be executed by a machine (see also Osterweil, 1987), because computers have precise execution semantics. Humans, however, do not. In fact, less strict formality rules allow flexibility in model creation as well as execution. With loose syntax and semantics, it is easier for the modellers to bring together multiple perspectives in a model. For humans, visuality and expressiveness are more important. (Curtis et al., 1992)

3.7 Two Approaches to Process Modelling – Software Process Modelling and Software Engineering Method Engineering

Now, having put process modelling in the context of software process engineering and studied the field of process modelling in more detail, the reader may have noticed similarities to software engineering methods engineering presented in the previous chapter. These similarities may be seen by comparing the 2.4 SE Method Engineering Architecture of Figure 2 in the previous chapter and the Software Process Modelling Architecture in Figure 4 above. Both have software engineering as their lower meta-level, and above it are the software engineering method engineering and software process modelling.

The purpose of this section is to clarify the relations of the concepts software process modelling and software engineering method engineering. For brevity's sake, they are denoted as SPM and SE ME, respectively. Figure 6 combines the two architectures mentioned and places the domains in the context of same architecture. It also makes some simplifications, which are more specifically addressed below.

In Figure 6, SPM and SE ME share the second meta-level. Below them, there is SE, and above them is the modelling method, similarly to Figure 4. The two domains on the second level both perceive SE, the former focusing on the entire set of phenomena on the SE level and the latter usually on a subset of these phenomena. Both of these are supported by SPM methods. The method specifies the entire SPM process producing a SP model and a portion of the SE ME process, a part which produces the process model depicting the dynamics of the method.

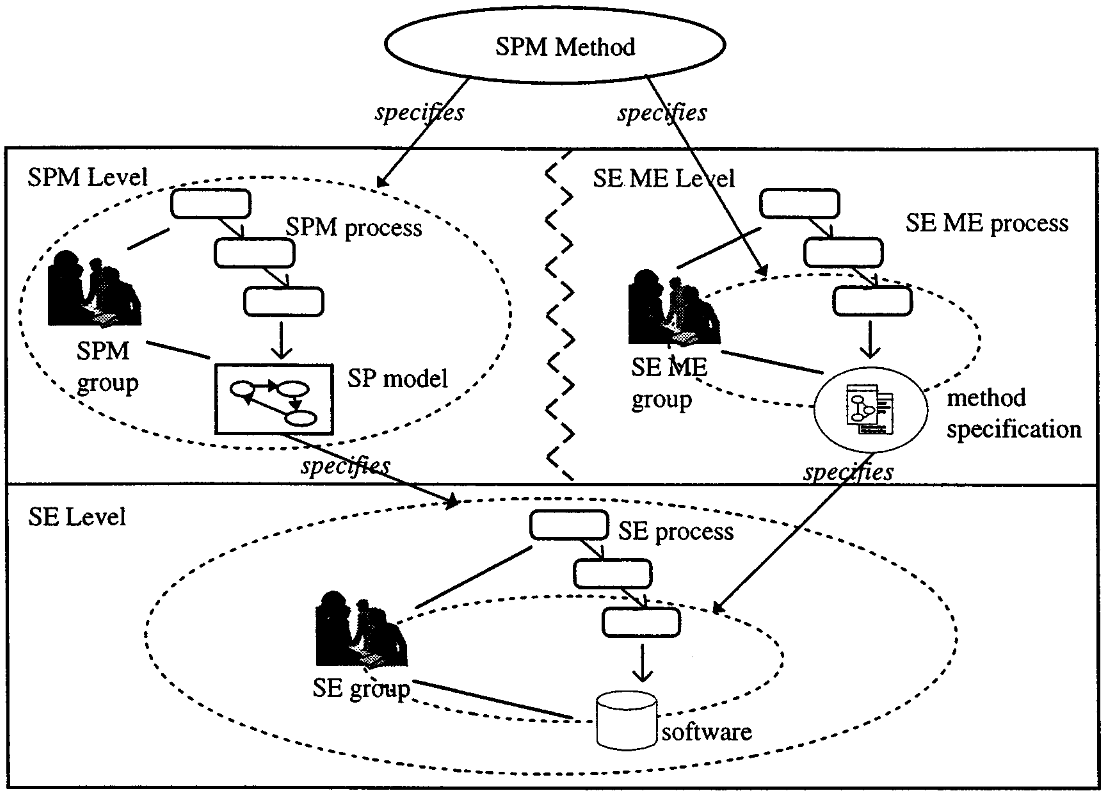


Figure 6. SPM and SE ME in Context

Now, we delve into the two phenomena located on the second meta-level in Figure 6: SPM and SE ME. The picture demonstrates an implicit categorisation of these two. On the bottom level of Figure 6, SE ME's focus is narrowed down by its technology-orientation. Its roots are in metaCASE research. By contrast, SPM's origin, the process movement, becomes apparent in its more explicit concentration on wider, conceptual, issues. This is intended to depict the main features in the current state of these fields, not to separate them, as the following paragraphs will show. That is to say, there is also non-technology-oriented SE ME research with a wider perspective to SE and PSEEs exhibit SPM study tightly bound to technology. We have, however, slightly simplified this state of affairs. For an example of a wide consideration of SE ME, refer to Burns and Dennis, (1985). A more limited one, again, can be found in Heym and Österle (1993), for instance.

Next, we examine the relationship of SPM and SE ME. Three aspects are brought up: the overlapping of the two domains, their differing scopes and, finally, the deviation in their level of abstraction and focus.

Overlapping. SPM produces a SP model and SE ME a method specification. As noted in the previous chapter (Section 2.4), a method specification is built of three parts, one of which is a process model describing the use of the method. The creation of this model conceivably involves process modelling. Let us illustrate this looking the matter from the point of view of the SPM method. SPM, possibly specified by a SPM method, is thus as a part of the SE ME process. In the picture, this is shown with the arrows from the SPM method to the SPM Level and to the SE ME Level. On the SPM Level, the SPM method specifies all that is needed to construct a SP model. The use of the method on the SE ME Level, again, relates only to a portion of the SE ME process and its product, namely, the process model included in the method specification and its creation. Thus, the domains overlap in that they both involve process modelling. This is represented with the winding line between the SPM and SE ME levels in the picture.

Scope. Traditionally, SE methods have only considered parts of the SE process, such as designing, whereas SPM, in the sense we use the term, has had a more comprehensive focus of the entire software process, covering the use of the methods in it as well. CMM's process definition demonstrated the relation of methods and process by placing methods as a subset of the software process (see the beginning of this chapter). This difference in scope has, however, began to blur as more comprehensive methods perceiving larger parts of the software process have emerged (see Harmsen, 1997, p. 5). Furthermore, PSEE-related SPM may well focus specifically on only a subset of the SE activities, finding itself on the same level of abstraction as the traditional SE ME.

Level of abstraction. A third difference between SPM and SE ME is in their level of abstraction. Although this distinction, similarly to the shift in scope, is no more than a coarse deviation in tendencies: SPM tends to see the SE process on a higher level of abstraction than SE ME. This is analogous to the deviating scopes. For example, if some design diagram, a software model, is produced using a method in the course of an SE process, the process model of the method specification might describe the detailed steps to be taken in the diagram construction while the SP model of the same SE process is likely to only depict the finished diagram as an end product and its construction may be seen as a single activity.

Focus. Relating to the difference in the level of abstraction, one more deviation can be found in SPM and SE ME. Seeing SE on a higher abstraction level, the former mainly focuses on the SE process and the latter, provided with a more detailed view of a part of SE, focuses on the SE products. Again, these features cannot be used to categorise the two fields to totally distinct classes as SPM depicts also the products and SE ME pays attention to the process as well.

To conclude, SPM and SE ME are overlapping domains located on the same meta-level. They mainly concentrate on different issues in the SE process and engage differing approaches to process modelling, emphasising slightly different levels of abstraction. SPM can be characterised as a broader, more abstract and process oriented approach, whereas SE ME concentrates on smaller parts of the SE process, stressing the SE products. Generally, it produces more detailed and product-oriented process descriptions to facilitate the method use. Nevertheless, the distinctions between the two fields are dim, and no explicit separation between the phenomena can be made.

Next, the final section of this chapter shortly summarises the number of aspects of software process modelling presented in this chapter, and presents need for a methodical approach to it.

3.8 Summary – Need for Method Driven Software Process Modelling

In this chapter, we learned about the role of software process modelling in a wider context, covering software engineering as well. Evidently, software process modelling is a multidimensional phenomenon, which is closely linked to software engineering and provides the basis for many other forms of process improvement. This is also expressed in the following quotation.

> Software development organisations are finding that they should first focus on defining the processes used in their software business, and only then select tools and methods to support these processes. [...] Effective use of defined processes requires that a process be documented, that personnel be trained in its application, that it gets performed as it is documented, and that its performance is enforced and measured. Once a process is defined in this manner, it can be deliberately and methodically improved. (Curtis et al., 1992, p. 86)

Software process modelling can be used for a number purposes. Process automation has received most of the attention in literature and process programming is prevailing. Recently, however, the importance of facilitating understanding and communication as an objective of process models has been acknowledged and a corresponding shift of focus is taking place.

The many purposes plausible for software process modelling have inspired the development of a multitude of process modelling languages. As opposed to modelling objectives, there exists no generally accepted classification of this bulk of PMLs. Since the use of various PMLs results in various models depicting various process perspectives with various process information, the choice of the PML to be used in a given modelling situation is dependent upon the intended uses of the process model. This stresses the crucial importance of goal setting.

To conclude, the young domain of software process modelling is only beginning to take shape and its inherent complexities seem well comparable to those of software engineering. This intricacy has imposed the need to study the modelling process itself and thereby emphasise the methodological aspects of software process modelling. Fundamentally, this involves merely adding another meta-level to the framework presented early in this chapter. More on this subject follows in the next chapter, which discusses the phenomena related to method driven software process modelling.

4 METHOD DRIVEN SOFTWARE PROCESS MODELLING

Definition. In chapter 2, (Section 2.3) we defined a software engineering method as a codified set of procedures the purpose of which is to guide the work and co-operation of the various parties involved in the development of software. In the SE Method Engineering Architecture of Figure 2, SE methods were developed on the ME level above the SE level. Furthermore, we cited Marttiin (1994) for his division of the method specification into three parts: a process model[1] describing the use of the method, a meta-datamodel covering the conceptual structures (the modelling language[s]) used in the method and an agent model illustrating the organisational environment of the method use.

The previous chapter, on the other hand, presented a Software Process Modelling Architecture (Figure 4 in Section 3.1), whose Software Process Modelling Level was dedicated to the software process modelling process executed by a software process modelling group and producing a software process model. In some of their activities, the modelling group used a PML. Which we commented to incorporate at least an implicit description of its use.

Now, a *software process modelling (SPM) method* may be defined as a codified set of procedures the purpose of which is to guide the work and co-operation of the various parties involved in the development of a software process model. This model development refers not only to the use of the PML, but more comprehensively to the entire software process modelling process. The PML use description thereby constitutes a more detailed subset of these activities. In Marttiin's (1994) notions, the PML notation is the SPM method's meta-datamodel and its use description a component of the SPM method process model. In the Software Process Modelling Architecture of Section 3.1, the SPM method was depicted in an ellipse highest in Figure 4.

[1] As noted earlier, Marttiin speaks of an activity model.

Next in this chapter, we present yet another architecture to bring software process modelling method engineering in the context of software process modelling (Section 4.1). Then, we outline some causality relationships between software engineering and software process modelling method engineering (Section 4.2). Thereafter, four existing SPM methods are briefly described with assets and limitations (Section 4.3). Finally, the last section (4.4) summarises the current state of software process modelling method research.

4.1 Software Process Modelling Method Engineering Architecture

To illustrate the relationships between software engineering, software process modelling and software process modelling method engineering, we devise a third architecture, which is constructed by combining properties from both the SE Method Engineering Architecture in Figure 2 and the Software Process Modelling Architecture of Figure 4. The resulting SPM Method Engineering Architecture is depicted in Figure 7.

The Software Process Modelling Method Engineering Architecture is divided into the following three abstraction levels: the Software Engineering Process Level, the Software Process Modelling (SPM) Level and the SPM Method Engineering (ME) Level. The first and second have already been discussed associated with the Software Process Modelling Architecture (Figure 4 in Section 3.1. Here, we focus on the second and the third as well as the relationship between them.

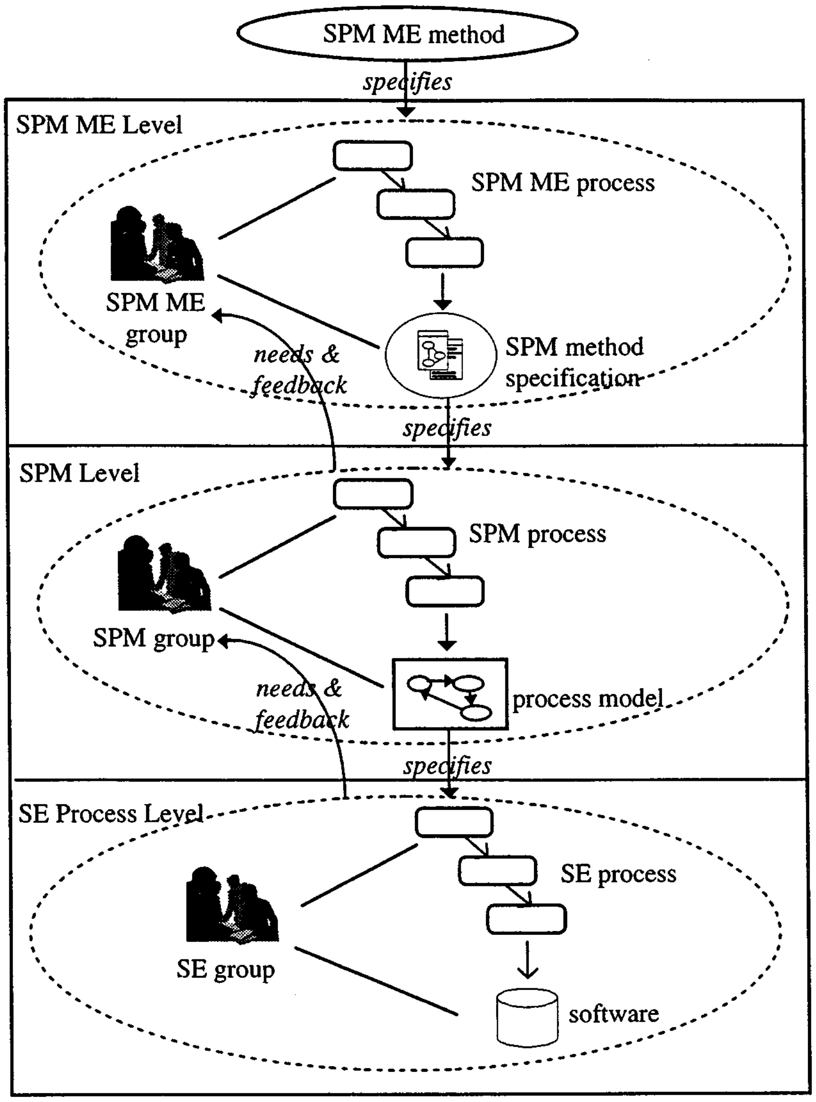


Figure 7. Software Process Modelling Method Engineering Architecture

SPM Level. On the SPM Level a *SPM group* executes a *SPM process* producing a *process model.* This modelling process comprises activities such as data collection, selection of PML(s) and the construction of a process model. The process is specified by the SPM method and associated PML(s) (not shown separately in the picture), both of which are developed and maintained on the SPM Method Engineering Level. The SPM group elicits the SE process needs and requirements for the SPM method and PML and addresses them to the *SPM method engineering (SPM ME) group.* They also give feedback concerning their use, as deemed necessary by Curtis et al. (1992, p. 79): "Feedback between language development and application experience is necessary to successfully evolve software process modelling techniques. Such experiments are crucial to support experience-based language design."

SPM Method Engineering Level. On the SPM Method Engineering (SPM ME) Level the *SPM ME group* executes the *SPM ME process* guided by the requirements and feedback received from the SPM Level, producing a SPM method (including a PML) specification for the SPM process. In practice, the SPM ME group may produce an adapted SPM method specification in the beginning of a modelling case, when the requirements for it have been collected. Later, when the modelling process advances to the process model construction, it is likely to intensively modify the PML to serve the modelling work. This is called situational method engineering.

The Software Process Modelling Method Engineering Architecture in Figure 7 adds an extra meta-level to the Software Process Modelling Architecture. This is the third level in Figure 7, the SPM Method Engineering Level, which is similar to the Method Engineering Level in the SE Method Engineering Architecture in Figure 2. On this level, the modelling methods are built. Thus method engineering in this architecture takes place one meta-level higher than in the SE Method Engineering Architecture. SPM method engineering level is the third, not the second meta-level. This owes to the focusing on software process modelling methods, not on software engineering methods.

In the following section, we contemplate a rationale for devising SPM methods by examining causality relationships between the meta-levels in the above architecture.

4.2 Rationale for SPM Methods – A Meta-Causality Scenario

Rationale. Software process modelling methods are to facilitate the management of the modelling process by bringing discipline to it, i.e., they attempt to standardise the modelling process. Thereby they are of use in assessment frameworks and improvement approaches (Section 2.7) using modelling as a tool, as the frameworks and approaches themselves do not pay attention to the practical "hows" of modelling. Conradi et al. (1993, p. 32), in fact, speak of an entire meta-process of which software process modelling is a part (see Section 3.2): "[...] we need methods to structure and control the meta-process. We cannot allow undisciplined access or transformations of meta-models and meta-tools." This replicates the pattern presented before in association with software engineering and software engineering methods (see Section 2.3) on a higher meta-level as was discussed in Section 3.1.

Proceeding from the lowest meta-level in Figure 7, the Software Engineering Process Level, to the highest one, the Software Process Modelling Method Engineering Level, the following scenario illustrates the repetition of some causality patterns in the architecture.

SE process diversity. On the SE Process Level, there exists no two identical software processes and thus the software process is unlikely to ever end up automated. "[...] there is no universally valid software process that can be automated once and for all, and then replicated for any new application development." (Armenise et al., 1993, p. 402).

⇩

SP diversity.

> As process descriptions are developed to meet process requirements which maybe expected to vary, clearly the design and implementation of these process descriptions should also be expected to vary. Thus it should be clear that there is no 'ideal software process description.'" (Osterweil, 1987, p. 10).

In other words, there is a need for various software process models (similarly to the need for various software engineering methods). This is exemplified by Yu and Mylopoulos (1994, p. 159): "The need for different types of software process models for different purposes may be compared to the need for different languages to represent software products at different levels [...]"[1].

⇩

[1] The authors refer to levels in the software process. The level products mentioned are thus requirements, design and implementation.

SPM process diversity. Process model diversity results in different modelling processes on the SPM Level. “In fact, it may well be that there is no universally applicable process modelling process, it is quite feasible that the right mix of different process engineering techniques needs to be configured for each specific situation.” (Kontio, 1995, p. 70).

⇩

SPM method diversity. There is no ideal software process modelling method which could be replicated to all modelling situations, but there is a need for various software process modelling methods. “A large repertoire of technologies is available, both for process and meta-process modelling and instrumentation, but a complete method base is missing.” (Conradi et al., 1993, p. 33).

Need for SPM method engineering. Stopping to modelling method diversity, i.e., the need for various SPM methods, the scenario above implies the need for SPM method engineering (SPM ME). The diverse methods suiting each modelling case are created by engineering the existing methods according to the case requirements and modelling objectives and by constructing new ones on the SPM Method Engineering Level. The situation is thus similar to that with software engineering methods: standardisation imposed by software process modelling methods and languages needs to be balanced with flexibility. This requires method engineering.

Extension to the scenario. An extension to the scenario is also quite feasible: the variety of methods to be developed might lead to different method engineering processes. This, again, could be controlled with method driven SPM method engineering, which leads to yet another meta-level. The SPM method engineering level is nevertheless the highest meta-level we reach in this research and any level above that is out of our scope.

Naturally, the scenario above is only an outline and provides a simplistic view to the multifaceted causality relationships resulting in the need for SPM methods. It nonetheless offers a practical illustration of the replicating patterns in the Software Process Modelling Method Engineering Architecture in Figure 7. As there is no universally applicable software process or a process model, there exists no uniform process nor a standard method for developing such a model, either.

Research focus. Method driven process modelling (and evolution)[1] has received increased attention during the last few years (see, for example, Conradi et al., 1993, p. 33; Madhavji et al., 1994; Madhavji, 1991, p. 240) and a few methods have already emerged (discussed in the next section). Conradi et al. (1993) list three topics that SE researchers are concentrating on:

- How can we describe a software process, in order to understand it and to facilitate communication and teaching of its rules and procedures?
- How can we better manage and improve the software process?
- How can we have humans and computerised tools co-operating in a co-ordinated and controlled way to support the software process? (Conradi et al., 1993, p. 26)

Stressing the first word, "how", the first item on this list indicates SPM method and PML focused research.

We found four PSEE-independent modelling processes that SPM method focused research has produced. We believe that these methods provide a sufficiently comprehensive overview into the current state of the research. They are briefly went through in the following section.

4.3 Existing Software Process Modelling Methods

Software process modelling has been studied for over ten years, but there are still less than a handful of SPM methods to guide the modelling work. Kontio (1995) characterises this state of affairs as follows.

> The software process field has largely focused on the study of different process modelling formalisms and the methodological aspects of process capture have received considerably less attention. Although the process models have had a central role in the software process domain since the 1980's, process capture has not been addressed explicitly until recently.

[1] Lonchamp (1994a) speaks of model driven meta-process support

> Even the software process workshops have made only brief references to the topic (Huseth and Vines, 1987; Ochimizu and Yamaguchi, 1987). (Kontio, 1995, p. 34-35)

Many of today's PSEEs (see Section 3.4) incorporate an implicit modelling method as well. These environments, however, aim at automating the software process, emphasising the enaction of the created model, and thereby deal with capturing the process in models only superficially. Their methods and PMLs, being hard coded into the technology, are often plagued with inflexibility. Furthermore, the modelling methods are granular and usually constitute a part in a more extensive meta-process as in EPOS (Conradi et al., 1994a), which includes abstract meta-steps to define and evolve process models. From the vast bulk of PSEEs and the surprisingly small number of modelling methods it can be concluded that the focus of research has obviously been on process programming and technology, leaving the methodological aspects with considerably less attention. Our focus here, however, is on more comprehensive SPM methods, which are independent of any specific PSEE.

The following four subsections each describe one currently available SPM method found in literature. First, in Subsection 4.3.1, we present Elicit (Höltje et al., 1994; Madhavji et al., 1994), which currently is the most comprehensive publicly available method for software process modelling. Secondly, the Descriptive Modelling Process developed at the SEI (Software Engineering Institute) is focused upon in Subsection 4.3.2. Then, Subection 4.3.3 discusses Process Description Approach (Klingler and Schwarting, 1994), and, lastly, a Quality Improvement Paradigm based modelling method (Kontio, 1995) is presented in Subection 4.3.4. Finally, we summarise some assets and limitations of these methods.

4.3.1 Elicit

Elicit is developed at the McGill University. It aims at eliciting formal descriptive process models from industrial software environments to facilitate understanding of the process and to thereby make process improvements possible. For more information about Elicit, see Höltje et al. (1994), Madhavji et al. (1994) and also Heineman et al. (1994). The method constitutes of the steps briefly listed below.

1. *Understand the organisational environment.* Identify and document the essential characteristics of the organisational environment.
2. *Define objectives.* Focus the modelling based on the documented context resultant from the previous step.
3. *Plan the elicitation strategy.* Among other things, select appropriate PML(s) and tool(s), allocate resources and schedule the modelling based on the previous steps.
4. *Develop process models.* Elicit process information, translate the process information into the process models and review the models.
5. *Validate process models.* Validate the produced models and have them formally approved by the recipient of the model.
6. *Analyse process models.* Analyse the process models in order to suggest process improvements.
7. *Post-analysis.* After the modelling the modelling method itself is analysed in order to improve it.
8. *Packaging of experience.* The experience gained from the modelling work is packaged for the future use of Elicit.

The current version of Elicit has been evolved in a pilot study and by applying it in two industrial scale projects (Höltje et al., 1994). Two tools have been used in these modelling projects; process information elicitation tool named Elicit, and a graphical model representation tool, STATEMATE (see e.g. Kellner and Hansen, 1989). As could be seen from the third step in the sequence above, Elicit does not predetermine any tools. Recently, the Elicit work has been continued by Turgeon and Madhavji (1996) towards supporting role-driven views in process modelling.

4.3.2 Descriptive Modelling Process (DMP)

The Descriptive Modelling Process (DMP) is developed by the Software Engineering Institute SEI, and it is based on SEI's modelling experience gained since 1987 (see e.g. Kellner and Hansen, 1989; Kellner and Hansen 1988) and it has been taught to software professionals for use in software process modelling (see Heineman et al., 1994, p. 510).

Currently, the DMP is described in an unpublished series of process definition guides such as Armitage et al. (1994) (for a more exhaustive list of these guides, see Kontio, 1995, p. 36). We did not manage to obtain a reference presenting a process description for DMP, but to our knowledge, the DMP starts with planning, management contracting, process familiarisation and goes on to an iterative modelling phase in which interviewing, model construction and verification and validation are repeatedly executed.

The DMP concentrates on the process information presented in process models including activities, artefacts and agents (Armitage and Kellner, 1994). This information is first extracted from the process documentation and interviews with process actors and subsequently modelled using freely chosen PML(s), resulting in descriptive models.

4.3.3 Process Description Approach (PDA)

The Process Description Approach (PDA) (Klingler and Schwarting, 1994) is developed in the STARS[1] program sponsored by the DARPA[2]. It includes two PMLs: the graphical IDEF0 and the textual Process Breakdown Structure. Together, they form a coherent whole describing activities, roles, work products, work product availability constraints and control constraints both graphically and textually

The PDA's process definition process is divided to three main activities which are further decomposed to the lower level of detail. Some recommendations for carrying out these activities in practice are also given in (Klingler and Schwarting, 1994). The main activities and their sub-activities are as follows:

1. *Process analysis.* Elicit process requirements and set process definition objectives as well as evaluate the requirements.
2. *Process model creation.* Develop different models of the process, choose the appropriate design and implementation alternatives as well as analyse the chosen process model.

[1] STARS = Software Technology for Adaptable, Reliable Systems

[2] DARPA = Defence Advanced Research Projects Agency

3. *Preparation for performance.* Create a process implementation and a project instance as well as schedule the process performance and train performers.

To the best of our knowledge, the PDA is being further evolved based on experience gained in its application.

4.3.4 Quality Improvement Paradigm and Experience Factory Based Modelling Process

In his licentiate's thesis, Kontio (1995) introduces a coarse-grained modelling process which combines the Experience Factory and the Quality Improvement Paradigm (QIP) discussed in Section 2.7. He argues that the steps of the QIP can be instantiated into a modelling process when the aim is to improve a software process.

The steps of the modelling process are as follows (Kontio, 1995, pp. 68-70):

1. *Characterise.* Obtain a realistic understanding of the modelling environment.
2. *Set goals.* Set the process modelling goals.
3. *Choose process.* Plan the process modelling, including the selection of notation.
4. *Execute.* The actual process modelling.
5. *Analyse and package.* The experience gained from the modelling process is collected and analysed to further improve the modelling process. Then, the modelling process is started again from the beginning.

Kontio's purpose for designing the modelling process is to combine his improvement-oriented process management framework, named Promises, and some selected assets from other process modelling work, namely, some PMLs and methods.

4.3.5 Assets and Shortcomings of the Methods

Regarding the current state of SPM method research, Avrilionis et al. (1996) state that

> Process models are difficult to construct, understand, maintain, make evolve, etc. This is due to the methods and tools currently available. Most of them are rudimentary because the process model domain is still in its infancy. (Avrilionis et al., 1996, p. 102).

Today's SPM methods mainly describe the modelling process in a life-cycle fashion describing *what* to do. They list the modelling steps thus offering general guidelines for modelling a software process. However, the SPM methods in our selection, possibly excluding the DMP describe only poorly, if at all, *how* the modelling should be done in practice, i.e., they are on a high level of abstraction. This is a comprehensive limitation characterising all the methods.

In addition, we paid attention to three properties of the SPM methods. These are: process context[1] focus, PML flexibility and SPM method flexibility. It was found that, in general, the first is appropriately taken into account, the second only partially and the third is totally neglected. The following three paragraphs shortly characterise the appreciation of each of these features.

Process context. Most of the methods duly appreciate the understanding of the process context. For this task, as well as for the collection of process information, they use a variety of data collection techniques, including document study, questionnaire and interview. PDA even utilises the participation of process actors.

PML flexibility. Another shared feature for all these methods, excluding the PDA, is that they flexibly select the PML(s) based on the process context. The selected PML(s), however, is (are) taken as they come, that is, the SPM methods lack any regard or means to modify it (them).

SPM method flexibility. This fixedness applies to the methods themselves as well. Most of them have obviously been defined only on a high level of abstraction for the sake of generity. Therefore, they always need to be specified, complemented with the how-part on a case by case basis, in accordance with the process context.

Flexibility of the PML and the SPM method itself are needed to dynamically conform the modelling process to the various requirements of each process context. This is achieved by engineering the methods on the SPM Method Engineering Level of the Software Process Modelling Method Engineering Architecture of Figure 7.

---

[1] The modelling environment where the modelling takes or is about to take place (Pèrez et al., 1996, p. 49).

Next, the final section of this chapter summarises the current state of software process modelling methods research and the need for them as well as discusses about the engineering the methods.

4.4 Summary – Need for Comprehensive and Flexible Methods

Software process modelling has been a popular research theme among the software process community for longer than a decade. In spite of that, the methodological aspects of the modelling have received less attention until the very recent years.

The need for flexible and modifiable methods and tools found in the field of software development is applicable to the software process modelling methods and tools as well. Software processes are complex and evolve continuously thereby imposing flexibility requirements on process modelling methods: it should be possible to adapt them for different modelling situations and to develop new ones in a controlled manner. The significance of a PML's fitting to the process context is promoted by Rombach (1990):

> The effectiveness of a tool in general and a process description language in particular, depends on the context in which it is used, the objectives it is used for and the degree to which its features are understood and used. (see Armenise et al., 1993, p. 403).

Currently, there are only a few methods dealing with the delicate issues of software process modelling. Their assets include paying adequate attention to the process context and process data collection. Additionally, almost all of these methods select the PML dynamically, based on the modelling context and objectives. On the other hand, their description of the modelling process is rudimentary and in a rather general level of detail. Furthermore, they do not take into account the need to modify the selected PML(s) or the SPM method itself. Therefore, situational software process modelling method engineering is required, as was illustrated above.

Summarising the current state of SPM method research, it can be noted that the state of affairs has progressed from that encountered by Madhavji (1994): "From all these experiences, one can conclude that currently there are no published methods on how to elicit process models form software projects (Madhavji et al., 1994).". Although some methods exist, there still is a marked shortage of context-free, comprehensive, flexible and context-oriented[1] modelling methods to guide the intricate modelling work. This flexibility and context-orientation necessitates method driven software process modelling and SPM method engineering.

Having climbed the meta-level ladder and reached method driven software process modelling, we now turn to examine the desirable characteristics of a SPM method. The next chapter introduces four fundamental software process modelling principles, which summarise the assets of the current methods as well as address their shortcomings.

[1] Context-orientation means taking regard of the context. Context-freedom, again, denotes applicability to multiple contexts.

5 FOUR FUNDAMENTAL PRINCIPLES FOR SOFTWARE PROCESS MODELLING

The previous chapters have discussed software process modelling background, and introduced the domain of software process modelling. Last, the need for software process modelling methods was expressed. In this chapter, basic principles for software process modelling are introduced. These are: flexibility, the use of multiple data sources, PML engineering and actor participation. (Sections 5.1 – 5.4). They are an attempt to combine the assets of the current SPM methods and to complement them with ill-supported aspects. Then, in Section 5.6, a modelling process is outlined to illustrate the application of these principles in practice. At the same time this process functions as an introduction to the practical part of our research in the next chapter, which executes the process thereby examining the principles in practice.

Presuppositions. There are two presuppositions for applying the principles in modelling. First, the modelling takes place in a maidenly environment, where the software process has not previously been modelled below the life-cycle level of detail. Second, the objective of the modelling is to facilitate understanding and communication of the process, which is a self-evident goal when modelling a maidenly process. These presuppositions have functioned as the main criteria also when evaluating the current methods and exploiting the best of them. They also label our PML engineering effort.

Method engineering and context-freedom. These principles are not intended as a method, but constitute fundamental guidelines applicable for modelling a software process. As guidelines, the principles leave space for situational adaptation, and do not specify how the modelling should be conducted in practice. They merely propose some issues to be taken into account. Thus, method engineering is required in their adaptation to a particular modelling case. To facilitate this, they are intended to be context-free: it is imaginable, that they are applicable to any maidenly process modelling context where understanding and communication of the process is to be facilitated.

The first principle, discussed in the next section, is the most universal one: flexibility.

5.1 Principle of Flexibility

Software process modelling should be adapted to the process context.

Context-orientation. The term process context is used by Pèrez et al. (1996, p. 49) to denote the environment of the modelling. "Because the environment forms the context of the model, it is referred to as the process context." Thus, the principle of flexibility emphasises the contextual relationships of the modelling and calls for context-orientation throughout the entire modelling process.

Present research. In Section 2.3, inflexibility was identified as a prevailing characteristic of the current CASE tools. Together with the neglect of organisations to assess the development context, it was regarded as the main inhibitor of CASE success. The same applies for PSEEs as indicated in Section 3.4. Situational method engineering fights this fixedness by constructing project-specific methods (see 2.4). However, unbending technology frequently waters down the method engineering efforts.

The SPM methods presented in the previous chapter (see Section 4.3) generally acknowledge the importance of adaptation of the modelling to suit the process context. This is implied by their emphasis on obtaining information of the context. Elicit begins by first understanding the organisational environment. The product of this step, documented context, is input to all subsequent steps of the method. Similarly to Elicit, the DMP contains a process familiarisation step and Kontio's modelling process has a characterise step. Process Description Approach, again, in its process analysis, gathers process requirements, models these requirements, sets objectives based on them and finally analyses the requirements. Apart from setting modelling objectives, the contextual expertise is utilised by most of the methods as they flexibly select the PML to be used in the modelling. This is supported by Elicit, Descriptive Modelling Process and Kontio's modelling process.

Flexibility is crucial for modelling. From the current SPM methods' emphasis on context study, it can be deduced that the issue is essential for successful software process modelling. The need for contextual information, again, is imposed by the desire to adapt the modelling process to each particular context. Accordingly, today's methods use this information to flexibly select the modelling objectives, through which the context affects the entire modelling process, including the selection of the PML. Thus, from the experience of today's SPM methods, it can be concluded that flexibility is crucial for modelling. As the intricacies of software processes vary from one to another, context-oriented flexibility on the part of the modelling process is profitable in that the objectives and requirements of the particular modelling situation are more closely met and its capabilities are more intensively exploited.

This requirement for flexibility is amplified when aiming at increased understanding and communication of the process, because such a modelling objective deals with the awareness of the process of the process context and thus necessitates intense bi-directional communication between the process actors and the modellers. This issue, however, is more closely examined below associated with the principle of participation.

Need for context data. The principle of flexibility, adapting the modelling to the process context, imposes a need for proper information of the context. Ensuring the provision of this information is one reason for introducing the next principle, the principle of using multiple data sources.

5.2 Principle of Using Multiple Data Sources

Data collection for process modelling should be based on multiple sources of data.

Present research. Using multiple data sources is an advice given in (Yin, 1994, pp. 90-99). Currently, most process modelling is also based on data obtained from multiple data sources (see, for example, Dandekar et al., 1996, p. 27; Pèrez et al., 1996, p. 50; Höltje et al. 1994). The most popular sources of modelling data are perhaps process documentation, questionnaires and interviews. Also the modelling methods discussed in the previous chapter generally mention more than one data source, covering the ones mentioned above, but they leave it up to the implementor whether these should be used at the same time or not.

Simultaneous use. Our emphasis is exactly on this simultaneity of using the multiple data sources. Used in parallel (i.e., to provide data to the same modelling) the sources can complement and validate each other, as Yin (1994, p. 92)[1] suggests. This renders the resulting process data more detailed, accurate and convincing. As the modelling process requires various data for different purposes, process data collected from diverse sources is in place. Namely, different sources are at their best in making different data. For example, process documentation is most convenient in contributing to the general understanding of the process and its context, providing the basis for the objective setting. Interviews, again, best provide more fine-grained information of the process, which is to be depicted in the model.

The effect of a maidenly context. In a maidenly process context the role of data collection is fortified, because in the absence of a detailed and up-to-date body of process knowledge such a body needs to be created. Only then can the modelling properly start. For instance, process documentation is likely to give fewer answers to the process modellers in a maidenly context than in a context formerly modelled. Furthermore, as the process actors are either not at all or variably familiar with "process-thinking", they may not understand process-related questions, or may misunderstand the nature and scope of the modelling activity.

These aspects introduce a challenge for process modelling attempting to facilitate understanding and communication of the process. With this objective, the purpose of the modelling is to lead the process actors to recognising their process in order to be able to improve it. Therefore, accurate and comprehensive process context information is needed for the modellers to steer the actors to the right track and to conform the modelling to their capabilities and needs. An evident and tangible way of conforming the modelling is through flexibility in regard to the PML used. This is the concern expressed in the next principle, PML engineering.

[1] Yin speaks of data triangulation resulting in higher quality studies.

5.3 Principle of PML engineering

PML(s) should be selected and modified in accordance with the process context.

The principle of PML engineering is really adapting the principle of flexibility to the modelling language. We have heightened it as a separate principle owing to the central role the PML used plays in modelling. The syntax and semantics of the PML should be sufficient for modelling a process in a way that meets the requirements of the process context and, on the other hand, the language should be simple and illustrative enough to facilitate understanding and communication of the process. Thus, in addition to the flexible selection of the PML, this principle suggests that it should be modified to suit the needs and abilities of the process context. This modifying may happen in the course of the modelling process.

The idea of PML engineering is simply to apply the method engineering experience discussed in Section 2.4 to software process modelling languages. This appears to be completely foreign to the current SPM methods (see 4.3.5 in the previous chapter) and even to the process modelling research as a whole.

Koskinen (1996, p. 92) notes that there are no PSEEs addressing customisability of process modelling languages. Her study considers adding PML customisability to a metaCASE environment to enable the building of process support for different projects (see also Koskinen and Marttiin, 1997). At that time, the approach presented in her study was unique in this sense. We also found that in spite of the obvious need for flexible PMLs as well as for the adaptation of existing ones, there have been few to pay attention to modelling method engineering. In addition to the approach presented by Koskinen, one recent paper addresses the issue. In their paper, Kaiser et al. (1996) introduce a "meta-linguistic" approach to modelling which allows to extend the PML which is embedded in a PSEE to conform to needs specific to a modelling case. Thus, current customisability approaches attend to the formal aspects of the PMLs rather than to user-related perspectives such as understandability and suitability of the language to the context of a particular use situation.

5.4 Principle of Participation

Process actors should participate in the modelling.

The participation of the process actors themselves in the modelling is a keystone in facilitating understanding and communication in a maidenly context. It is difficult to imagine a more effective communication occasion than the actors and the modellers discussing and evaluating the details of the process model together. This would definitely increase the actors' understanding of their process.

Prerequisites of participation. Actor participation lays further flexibility requirements on the modelling. Such intense communication is not possible if the participants have no say on the modelling objectives and the expedients used or their needs are not reacted to. Therefore, the form of participation and the PML have to be selected and the PML may also have to be modified according to the actors' capabilities, needs, wishes and requirements to enhance motivation and communication. As noted above, this amplifies the significance of contextual data obtained through data collection.

Requirements for the PML. Basic requirement for the PML is visuality. Swenson et al. (1994, p.13) list requirements for a visual language in a collaborative modelling setting. Among other things, they insist that the use of the PML should be easy enough to be used by the actors, the process descriptions should match the actors' perception of the process and present the process on the same level of abstraction that each of them work. In addition, the PML should allow parallelism and iteration of activities.

Participation in the software process modelling domain. As was the case with the principle of PML engineering, actor participation is largely neglected by the current SPM methods as well. In general, the methods include the actors in the data collection phase (interviews and questionnaires), but only Process Description Approach speaks of process performers participating in the actual process modelling. Generally, the methods consider it is possible to use process actors in the verification and validation of the models.

Experiences of participation. Höltje et al. (1994) express the value of the process actors' domain knowledge in saying that they "[...]faced many domain related difficulties which were solved with the help of domain experts". This implies the value of actor participation. Accordingly, in their discussion of other disciplines' possible contributions to software process modelling, Totland and Conradi (1995) argue that our field could learn from concurrent engineering by employing user participation in building process models. Moreover, Swenson et al. (1994, p. 2) demonstrate the participative scheme working in practice in the field of business process re-engineering. They use the term collaborative planning to denote a "strikingly different approach" in which "[...]process plans [models] can be created and modified by the end user allowing users to experiment and find the best process." Actor involvement such as this goes, in fact, beyond participation, and is perhaps the ideal situation.

Being under constant time-pressure, however, the resource scarce software engineering industry is unlikely to absorb such an intensive process modelling effort. In a neighbouring field, information systems development, a milder approach is being applied as well. There, participative system design (see Hirschheim and Klein, 1992, p. 301) has received considerably attention, as is reported by Iivari (1996, p. 97): "The relationship between user participation and IS success is one of the most investigated topics in IS research."

For example, the Scandinavian Collective Resource Approach (CRA) (see Ehn and Kyng, 1987) and the ETHICS method (see Olerup, 1989) apply a socio-technical approach to the design of computer assisted work One of the main assumptions of the socio-technical design is that end users should participate in the design (see e.g. Avison and Wood-Harper, 1990, p. 130; Ehn and Kyng, 1987). Olerup (1989, pp. 61-62) lists three forms of participation. These are (i) consultative participation in which end users are consulted but the decisions are made by designers, (ii) representative participation involving representatives of the end users in the design and finally, (iii) consensus or democratic participation aiming to engage all the end users throughout the entire design process.

Summarising participation. To conclude, the benefits software process modelling stands to gain from actor participation in modelling are obvious. Associated with the above principles, it is indeed an asset in the modellers' toolbox, especially when the process has not been previously modelled and the aim is at facilitating understanding and communication of the process. Yet it has received attention only in the neighbouring fields of science. The convincing experience obtained elsewhere is a solid ground for arguing that the principle of actor participation should be applied in process modelling as well. Involving the actors, experts of their own processes, results in more accurate and complete process models.

5.5 Relationships Between the Principles

The four principles, flexibility, using multiple sources of data, PML engineering and participation are distinct yet interrelated. They represent aspects of software process modelling that we consider important when the modelling takes place in a maidenly context and aims at facilitating understanding and communication of the process. The first and the last principle, flexibility and participation are the most significant ones. They are closely tied in that flexibility of the modelling process renders the participation of process actors in the modelling possible. The other two principles actually apply or serve these two. The principle of PML engineering is applying flexibility in practice and the principle of using multiple data sources is to provide accurate and comprehensive process information and contextual data to enable flexible conforming of the modelling process to the context.

The remainder of this chapter deals with applying this conceptual construct in a modelling process.

5.6 The Principles in Practice – a Software Process Modelling Process

An example application. The principles introduced above provide only general guidelines, proposing issues to be taken into account in modelling a software process in a maidenly context to facilitate understanding and communication. To apply these guidelines in practice, they have to be realised in a sequence of activities. Therefore, the purpose of this section is to outline a modelling process utilising the principles. It provides an abstract design for the modelling with a limited set of suggestions for implementation. The actual implementation of the process is the focus of the next chapter.

The process steps in Figure 8 build up one plausible adaptation of the principles to use in process modelling. This process is based on our limited experience of process modelling and we deem it imaginable that any number of processes demonstrating such an application could be devised. This process, however, was found to be implementable in our setting, and is seen as such in any modelling situation similar to ours, which is discussed in the next chapter.

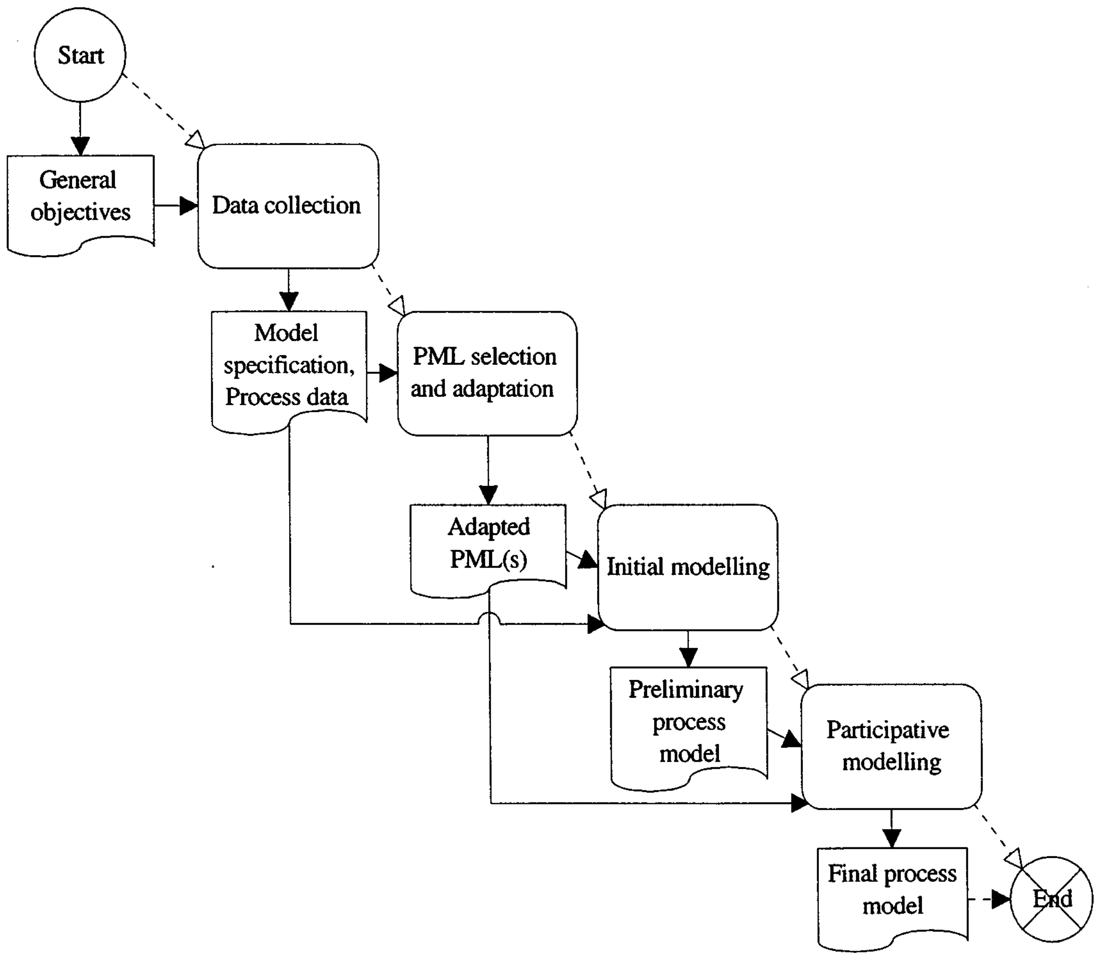


Figure 8. Process Modelling Life-Cycle

Although the process is presented as a linear sequence in Figure 8, this is only intended to demonstrate the flow of intermediate and final phase products in the process, not to specify a strictly sequential order of the steps. On the contrary, overlapping of distinct steps may occur in abundance.

Facilitating understanding and communication in a maidenly context. The presuppositions of this modelling life-cycle are those of the principles: the software process has not been previously modelled and the objective of the modelling is to facilitate understanding and communication of the process. No means to elicit the validity of these in a given process context are given, but they are used as a starting point for the modelling.

CMM and Experience Factory (see Section 2.7) aim at improving the software process based on experience collected of process execution. Accordingly, Pohl (1996, p. 51) is one to state that process improvement can be initiated by recording experience gained during process execution. On the other hand, he claims that any process improvement requires explicit process models (p. 59). Consequently, we need explicit process models recording process execution experience, i.e., *descriptive modelling*. This enables the process actors to see the internal structure and interdependencies of their process' tasks thereby facilitating understanding and communication. As shown in Section 3.3, this objective is generally perceived as the modelling objective functioning as a foundation for others – it is the first milestone on the path to experience-based process improvement.

The objective of facilitating understanding and communication is the abstract goal of all activities in our modelling life-cycle, but the actual focus is more practically on characterising and describing the process, on making it visible to its actors. Therefore, increase in understanding is not measured in any way, although it would be possible with carefully planned questionnaires or interviews both before and after the modelling process execution.

A notable omission in the modelling process is the exclusion of the analysis and further development of the resulting model and the packaging of the modelling experience, which are not covered in the process. They were considered redundant from the standpoint of facilitating understanding and communication.

The following subsections briefly discuss each step of the modelling life-cycle in Figure 8. They illustrate the utilisation of the principles in a lower level of abstraction thus leading us closer to implementation.

5.6.1 Step 1: Data Collection

This first step is an application of the multiple data source principle (Section 5.2). Its purpose is thus to gather data from multiple sources to serve the remainder of the modelling process: process context related information to enhance flexibility and factual process data for the modelling itself. This necessitates the execution of three tasks:

1. Process context study. The very first thing to accomplish is to obtain sufficient understanding of the process context to be able to decide how the rest of the data collection and the modelling process should proceed.
2. Collecting model requirements data and process data. This is the main task of data collection. Model requirements data denotes information on the wanted features of the process model: level of abstraction, types of data depicted, etc. Process data refers to the process information to be depicted in the model.
3. Elaboration of the collected data. After the actual data collection, the data is documented so as to be usable in the later in the modelling. This involves producing a model specification document and a process data report.

Yin (1994, pp. 78-102) lists six data sources to be used in case studies: documents, archival records, interviews, direct observation, participant observation and physical artifacts. No case of process modelling research has reported the use of participant observation, but we find it usable in the first task, possibly also the second. This is an ethnographic[1] technique in which the modellers themselves work as actors of the process to be modelled (see Yin, 1994, p. 87).

[1] Ethnography has a long history in social research (see Schwartzman, 1993) but it has also been used for systems design (Hughes et al., 1994).

Process documentation study, again, may be of use in both of the first tasks. To obtain deeper insight into the software process itself than paper documents can offer, interviews and questionnaires may be used, possibly also studying the actual software produced or other process products. Additionally, actor participation in the modelling is another technique supporting the first two tasks of data collection, as discussed above in Section 5.4. All these data sources are used to complement and validate each other.

This step results in a process model specification document and a process data report.

5.6.2 Step 2: PML Selection and Adaptation

This step is a direct outcome of the principle of PML engineering (see Section 5.3). Its purpose is to select a modelling language deemed appropriate based on the understanding of the process context and the model specification. The PML is also to be moulded to adapt to the particularities of the modelling situation at hand, without forgetting the requirements imposed by participative modelling. In this step, three distinct tasks can be identified:

1. Survey on existing PMLs. One or more existing PMLs might suit the process context. Therefore, they need to be surveyed and evaluated.
2. Selection of the most suitable PML(s).
3. Modification of the selected PML(s) to fit into the modelling situation. At this point, the PML engineering may include any changes to the selected language.
4. Testing of the resulting PML(s). Finally, the conformance of the PML is tested in a simulated modelling situation. Real process data may be used in the modelling.

In the second task, a form of evaluation framework may be employed. This should take into account the significant features of the process context. It could be based on the model specification document. When the PML has been selected, an explicit SPM method engineering process and language might be utilised to mould it.

The ideal outcome of this step is a PML directly applicable to the modelling situation. Still, if need be, small changes to it may be made also later on during the modelling process, provided that they do not impede understandability of the process models.

5.6.3 Step 3: Initial Modelling

The third step, initial modelling, is to provide the following participative modelling with a wieldy software process model to work upon. Based on the process data and obeying the model specification the modellers create this preliminary model, which depicts the software process as they see it. Then, the resulting model is verified against the model specifications.

5.6.4 Step 4: Participative Modelling

This final step is a realisation of the participation principle. While involving the process actors in the modelling, it also makes up a data source in addition to those used in the first step, data collection. As mentioned in Section 5.4, this involvement aspires to obtaining more accurate and complete information of the process under scrutiny.

Participative modelling takes the preliminary model constructed in the previous step as an input and further elaborates the process model to its final state using the PML(s) previously selected and adapted.

The step is composed of the following tasks:

1. Preparation. First, the participation needs to be planned. This includes such issues as selection of the manner of participation, motivation of the actors, informing, scheduling, model representation etc.
2. Participative modelling. The actual participation with model editing and possible small PML changes.
3. Assessment of the depicted software process. Here, the fruits of the modelling are collected. As noted in the beginning of this chapter, the desired facilitated understanding and communication are not measured in any way. This task, however, is based on the supposition that the modelling was of use for this purpose, and collects the actors' thoughts of how the process might be improved.

4. Assessment of the modelling process. The very last task of the modelling process is to evaluate the process itself. This aims at finalising the modelling experience with the actor's comment on its validity. With the actors appreciating the effort as a whole, future modelling in the same context could be based on the results of this process, including the contextual knowledge, the process model, some techniques etc.

Feasible ways of implementing actor participation include modelling sessions or walk-throughs in which the actors, participating in groups, may be given the possibility to comment and edit the preliminary model. Therefore, the model needs to be represented in a magnified form to be seen by all the group members. The assessment, again, might take place in a brainstorming where the actors generate improvement ideas for their software process. There are, of course, a number of degrees for actor participation. The degree of involvement should be attributed to the modelling situation.

Note that there is no separate step for model assessment. Such a step is deemed unnecessary owing to the actor participation. It is assumed that model correctness, completeness and consistency are optimised already in the participative modelling, that is, the model is assumed to be correct, complete and consistent enough to facilitate understanding and communication.

5.7 Summary

This chapter introduced four fundamental principles for modelling a software process in a maidenly context to facilitate understanding and communication. Combining the assets of the current SPM methods and adding some complementary issues, they are guidelines proposing issues to be taken into account in different phases of software process modelling.

First, the principle of flexibility is the most pervasive one in that it gives a general guideline to be applied to the entire modelling process. Second, the principle of using multiple sources of data focuses primarily on the data collection for modelling insisting simultaneous use of multiple data sources to enhance correctness and completeness of the collected data. Third, the principle of PML engineering specifies the application of flexibility for the PML's part by suggesting that the modelling languages should be modified to conform with the process context. Fourth, the principle of participation involves the process actors to increase the model quality.

The previous section demonstrated adaptation of the principles to a modelling process, constituting a rudimentary design of a modelling life-cycle. Some example techniques were also mentioned to illustrate the implementation of the life-cycle in a real-world modelling situation. These included two techniques largely neglected in current software process modelling literature: participant observation and actor participation. The first may be used to obtain information of the process context and the second is intended to function both as a data source and as a communication event to facilitate understanding and communication of the software process.

The purpose of the next chapter is to present an implementation of the outlined modelling process in a real-world context. It provides an experience report to respond to the need expressed by Starke (1993, p. 15): "[...] the software process community lacks experience in modelling real processes. Languages, terminologies and meta-models have to applied to industrial problems to gain such experience and to improve applicability." We hypothesise that the principles be applicable to any modelling situation similar to the one under scrutiny in the next chapter.

6 MODELLING CASE

The purpose of this chapter is to present our experiences in applying the principles of the previous chapter in practice.

The modelling case began with reading background research literature on process modelling including PMLs, and the process documentation. At the same time, we observed the process context to be able to implement our modelling process in a way suitable for it. Soon, the interviews were undertaken. After four interviews, we conducted a small-scale modelling case testing our techniques and the PML for the initial and the participative modelling in practice.

In parallel and after this pilot modelling, the rest of the interviews and the PML selection and modification were proceeded with. This PML engineering continued until initial modelling, which started after the intensive data collection was finished. Some supplementary process data was collected in the initial modelling phase as well. Once an initial software process model had been constructed, participative modelling, the final step of the modelling process was launched.

All in all, the modelling life-cycle steps presented in the previous chapter have been intertwined into each other in practice. Especially the first and the second steps, data collection and PML selection and adaptation, were carried out in parallel with others. In the following, the modelling case activities are unfolded in a logical order, which does not quite match the chronological one presented above.

To begin with, the next section (6.1) provides an overview of the process context. Then follows a description of the small-scale pilot modelling process we executed at the beginning of the data collection (Section 6.2). Next, Section 6.3 briefly discusses the issue of modelling tools, which was largely considered to be out of the scope of our research. After that begins the actual modelling process.

First, the data collection experiences are presented (Section 6.4) and then Section 6.5 presents the initial modelling in its brevity, before turning to PML selection and adaptation (Section 6.6). This PML engineering was actually intertwined with the data collection, the pilot modelling and the start of the initial modelling. Further, Section 6.7 presents our participative modelling experience, which concludes our modelling case. Finally, the modelling experience is shortly summarised and its outcome discussed (Section 6.8).

To properly understand the modelling case, the reader must be familiar with its context. Therefore, it is presented in the next section.

6.1 Process Context

We conducted our study working for Nokia Telecommunications under a temporary contract. The software process, which we modelled was being performed by a single software team in Nokia Telecommunications' PMR-unit. Both of us had previous experience of the team; we had been working in it for some months before the study was started. The following subsection presents our employer and the main issues in its history of software engineering and in the subsection after that, the software team under scrutiny is presented.

6.1.1 Nokia Telecommunications / PMR

Nokia Telecommunications / Network Systems / PMR-unit (Professional Mobile Radio) develops and produces private mobile radio systems. Among other groups, government officials, large industrial plants, and transportation companies are significant customers. At the time of research, Nokia was developing its first standardised digital PMR-system which is based on the TETRA (Trans-European Trunked Radio) telecommunications standard established by ETSI (European Telecommunications Standards Institute). The TETRA standard is the first digital PMR standard in the world, and the need for it among European authorities was considerable.

Nokia has always concentrated on product quality and customer satisfaction. Nokia Research Center has studied software process management and control - among other software engineering issues - to provide support for each of Nokia's software development units. Nokia has ISO 9001 certificate, which forms the basis for Nokia's software quality. The company has also produced and uses additional documentation not required by the certificate. The most important of these is called Product Process Manual, which describes the entire process engaged in producing a product such as the TETRA system. The software process we focused on was located inside this product process as a subprocess among others.

6.1.2 Software Team, Tools and Process

The software team under scrutiny consisted of more than ten highly motivated, competent and committed members. At the time of research, it had only existed a little longer than one year. As a result, most of the members had only recently entered Nokia. In combining fixed and open elements of teamwork, the team was organised in a manner comparable to the *structured open team* discussed in Constantine and Lockwood (1993). Internally, the team was relatively autonomous in its action, which was customary in PMR.

The team is specialised in real-time embedded system development, an area considered to be one of the most challenging in software engineering. Most of the designers in the team use a novelty graphical CASE tool called SDT which is integrated into the software process. The tool is used in the software production process phases from specification to testing. Some designers involved in hardware related programming use more elementary tools. Much of the testing also requires the use of hardware related appliances. During the research, the team worked under a tight time schedule to develop the new software product, a project in which virtually all of the members were involved.

The software process of the team was rather implicit and changes were made dynamically i.e. the process was changed when problems occurred. This was due to a number of reasons: the software team was relatively young and had many new members, a new comprehensive tool had recently been adopted to the process, and the current project imposed a considerable time pressure. The characteristics above are also reasons for the fact that there were no history records of previous projects, only some produced by the current project.

The need for making the software process more explicit was recognised by the team we worked in. They understood the value of a process model as an expedient to ultimately improve the software process. Hence, we had few problems motivating our colleagues to ponder the details of their process and help us picture it in a graphical representation. The difficulties lay in the implicitness and complexity of the process itself.

Our first move to deal with the process was to test out ideas of modelling in practice before setting about the actual data collection. This is the theme of the next section.

6.2 Pilot Modelling – Prototyping our Modelling Process

Practically all of our modelling process was executed for the first time. The software team members had not taken part in such an exercise until then, but also we were inexperienced in conducting a field study. Therefore, we conducted a pilot study of practically everything we planned to do. This pilot modelling began after four interviews with the objective to experiment with the selected techniques in practice.

Piloting in case study research. The piloting was primarily inspired by Yin (1994, pp. 74-76). He suggests that a case study should be piloted when preparing for data collection in the research. Thereby, he argues, the researchers can refine their data collection plans with respect to both the content of the data and the procedures to be followed.

Prototyping in software engineering. By the same vein, software engineering uses prototyping is suited for dealing with changing customer requirements: if the customer is uncertain of his or her requirements, a prototype of the software is first created to afford him or her a concrete basis to specify the wished software product properties. Similarly, our pilot modelling case comprised all the same activities the proper modelling case was to include, thus prototyping our scheme.

Process. The pilot modelling comprised the same activities as the proper modelling case, but in a smaller scale. The study began with interviews, then an initial model was created with the selected PML, which was modified in the course of modelling. Finally, the model was further developed according to the comments of the participating process actors. This prototype-modelling was undertaken as soon as we had managed to find a suitable PML and to collect enough process data. The entire pilot modelling was carried out well before the interviews were finished. After the interviews, the resultant model was updated according to the requirements arisen in the interviews, and stored on a file server for all the actors to familiarise themselves with and to comment upon. This, however, turned out to be unnecessary, as many of the actors commented the wall-diagram version of the model and the electronic model did not catch anyone's attention, at least not to our knowledge.

Pilot process model. The pilot process model had two abstraction levels. The upper level, created first, described the entire process we were to model. Essentially, it was the same as the current official process model represented with our PML. The lower abstraction level, again, was founded on three interviews and described two of the upper level activities in more detail. We added a third abstraction level as well, but it was discarded by the process actors as superfluous.

Communication model. Apart from the actual process model, we constructed a separate model depicting communication related to the two activities modelled on the second abstraction level as well. This model was connected to the lower abstraction level process model through one symbol. The rationale for creating such a model originated from our observations and the interviews: especially the newcomers to the process had indicated that communicating with more experienced process actors had been their primary source of information at the beginning. Nevertheless, it later turned out that albeit the process actors perceived communication important, they saw no reason to model it. This relates to the resource modelling discussion presented in the next section (see 6.4.7.3). Although the communication model contained essential information about the process' dynamic action, it was not worth the enormous updating required due to fluid changing of the data it attempts to grasp. In a small process, it was argued, such a model does not contribute the actors' understanding of their process, since the information is nonetheless readily available to them.

Experiences. Via piloting we managed to eliminate the most flagrant mistakes like asking too many questions in the interviews or using a superfluously rich notation in the models, which otherwise could have cost us a considerable amount of time to repair. Furthermore, we were able to perform practical contemplation to shape the implementation of our plans. We could sharpen the thoughts by creating a prototype of the implementation. Resting on the experiences gained during the pilot modelling, we were able to repeat the same pattern in the proper modelling. The PML had been well understood by the process actors and participative modelling had served its purpose: the model was refined. Even though using the tools was laborious, they were usable, and the actors gave us positive feedback upon them. The pilot modelling process also affected the planning of our interviews by rendering us a clear view of what kind of information we would need. This directed the interviews to a more structured direction, which was found slightly problematic, as discussed below (see 6.4.5.4.3).

Usability of piloting experience. Although the pilot modelling proved that our modelling scheme would work in practice, it provided only a simulation of the actual modelling. Therefore, as the following sections will show, further adjustments were to be made. Nevertheless, much of the piloting experience and some of the products were directly usable in the actual modelling case. As interviews were the subject readily at hand, they benefited most from the piloting. When the latter phases of the modelling process, especially participative modelling, were reached, we had learned more about the process context and had to conform the previous experiences.

During the piloting we also picked and tested the tools for our modelling. This issue is shortly discussed in the next section.

6.3 Tools

Modelling tool selection and adaptation was conducted with little effort in parallel with the pilot interviews and is considered to be of minor importance. To keep within our main subject, we did not carry out any comprehensive tool information gathering and tool selection procedure, but simply picked the first ones that were found suitable for our purpose.

All the process model diagrams were created first using a computerised drawing tool supplemented with the symbols of our PML (see Appendix H). Subsequently, the computerised diagrams were presented with the so called wall diagram technique utilising coloured cardboard symbols and woollen yarn attached to a large sheet of cardboard with sticker (Appendix I). The only purpose for using wall diagrams was to afford collaborative modelling with diagrams large enough so that a group of people could easily discern even the smallest of details. Any representation large enough to fulfil this goal would have been appropriate. The advantages of the wall diagram technique are its clarity, visuality and scalability to the convenient size. Disadvantages include inflexibility in the form of laborious editing and large wall space requirements.

Now, the course of examination leads us to the experiences gathered in the first step of our modelling process, data collection.

6.4 Data Collection

The structure of this section is as follows. First, the objectives of data collection are presented (Subsection 6.4.1). Next, our data collection process is superficially examined to illustrate to the reader the relations of the methods and the resultant data (Subsection 6.4.2). Finally, all the methods we used are discussed in more detail with the associated experiences gained (Subsections 6.4.3 - 6.4.6).

The reader should keep in mind that each of us had familiarised himself with the software process under scrutiny during a short-term employment before the study. This previous engagement is not included in the discussion below, although that period unarguably contributed a great deal to our initial understanding of the context.

6.4.1 Objectives

The objective of the data collection was to obtain information regarding the modelling context. This relates to the principle of flexibility discussed in the previous chapter. More specifically, our intent was to establish two categories of requirements and to collect process data. The requirement types were modelling requirements and model requirements.

The *modelling requirements* were to answer the question "How should the modelling be carried out?", which can be refined to questions such as: "How busy are the process actors?", "What kind of support can be expected from the management?", "How interested are the process actors in modelling the process?" and "What kind of things are they interested in modelling?". Modelling requirements data was needed to decide how we should proceed with the process modelling.

The *model requirements* can be illustrated by the question "What kind of data should be represented in the model and how?" which may be refined into such questions as "What sort of problems seem to appear in the process?", "What determines the changes that occur in the process?" and "What kind of data do the actors themselves want the process model to depict and what kind of concepts are they used to dealing with?". Furthermore, collecting model requirements data entailed finding cues to deduce the appropriate level of detail and formality of the process model.

6.4.2 Overview of Data Collection

In data collection we adapted Yin's (1994, pp. 90-99) three principles of data collection into our case. The principles Yin presents are the use of multiple sources of evidence, the creation of a case study database and the maintenance of the chain of evidence. The first principle was realised by using multiple sources of data, as our principle suggests (see Section 5.2). The second principle took shape in our case database for the collection and organising of the data collected, such as document summaries, miscellaneous observations, interview reports etc. Finally, we maintained the chain of evidence by recording our findings and referring to them later in our study.

The multiple sources of data used were: participant observation, process documentation study, interviews and querying. Participative modelling, constitutes a data source as well, is dealt with in its own section, 6.7.

The very first task in our data collection was to open our eyes and ears to all the events occurring in the software process. We started to observe the actors and recorded all observations perceived as significant in a database for later review. At the same time, the process documentation study was undertaken. Observation continued throughout all of our study, albeit with considerably less effort later on than in the beginning.

After nearly a month, we started planning our interviews. An outline of the interview questions and a vague idea of the course of a semi-structured interview were further specified and developed during the first interviews. Therefore, the first few interviews may be called "pilot" interviews. This development continued until the end of the interviews, although the changes made were of ever diminishing significance. Consequently, no two of the interviews were completely identical.

Unpacking the interviews, which we carried out in parallel with the interviews themselves, was a notable task in its own right, although no substantial technical or methodological innovations were made. We produced clear written reports of all the interviews to be used as a basis medium in the extraction of the various data discussed above. After all the interviews had been finished, we sent the interviewees an e-mail query with some supplementing questions. The results of the query were added to the bulk of data resulting from the interviews.

At the end, modelling requirements were extracted from the interview results and a corresponding model specification document was written. After this specification had been approved by our management, a more structured process data report could be produced to offer a basis to build the initial software process model on. At this point, the interviews were supplemented with ad hoc inquiries conducted among the process actors.

Since the data collection phase was the starting point of our modelling case, we also had to pay attention to the motivation of the process actors during the data collection phase. This delicate task was tended mostly by distributing information concerning our chores, their meaning and progress via e-mails (see Appendix A) and in regular group meetings. No severe difficulties were encountered. In fact, the actors became ever more genuinely interested in our doings and some even came forward with their own ideas and theories regarding the software process and software process modelling.

To sum up, we used a whole variety of methods and were thus able to collect diverse experiences. Going into the process described above, the following subsections present all the data collection methods along with the associated results and experiences. To begin with, the process documentation study is discussed (Subsection 6.4.3). The second subsection (6.4.4) deals with participant observation and, finally, the particulars of the interviews are presented and discussed in Subsection 6.4.5.

6.4.3 Process Documentation Study

Our process documentation study comprised the following documents: the current project plan, the existing process description, the product process description, the operational manual plus various quality system documentation and implementation instructions. The essence of the documents was extracted into our database for later review. The following subsection discusses the vital content of the most significant documents and their role in our study in more detail.

Finding the documents. Owing to our previous engagement with the software process under study, we already had a vague idea of what kind of documents to look for. This idea was complemented with information from the project manager and a quick walk through the documents located on the network file server where the known documents were stored. Our access to the documents was unlimited due to the work contract we had made.

The role of the documents in the software process. The role of process-related documents in the target process was minor. Generally, they were considered to be management tools and practitioners might not even know that some document existed.

They had been written mostly for administrative purposes for different levels of management: project management, product program management etc. Some of them functioned as communication mediums inside and also outside the organisation. A few of the process-related documents were used as an introduction to the software process for newcomers.

6.4.3.1 Description of Essential Process Documentation

Project plan. As is typical for such plans, the plan of the current major software project contained a description of the project organisation, resource allocation to various responsibilities and the project's division into phases along with tasks and time schedule. Project organisation and task allocation information were useful in deciding the type of information different persons might possess and their roles in the process. The project plan was instantiated from the software process description, which provided us with some conception of the relations of the two documents.

Software process description. Previously, the software process had been described as a simple waterfall. The description contained a textual narrative of the phases and lists of tasks in each phase. The description was to function as an abstracted characterisation of a software project. It clarified the base to build our modelling on, and thus set the starting point for our modelling.

Product process description. The software process was actually a subprocess in a more comprehensive product process covering all the processes required to build a fully functional telecommunications terminal. The software process description was in a direct relation to this more general process through product process milestones – synchronisation points between all the sub-processes. The product process model description offered us a practical oversight over the interfaces of the software process by showing its role in the development of a terminal product.

Operational manual. This general document contained the description of the basic processes and principles of operation in the PMR business unit from business idea to support functions. This was the public document that could be delivered to the suppliers and customers interested in PMR. From our standpoint, the document contained a general all-embracing description of the wider context of the software process.

6.4.3.2 Role of Process Documents in the Study

Process documentation study proved useful in obtaining an initial picture of the software process. The process being already documented to some degree, we could utilise this portrayal in our navigating to the essential process information. The general level of the documentation, however, inhibited us from further using process documentation in our study, but we had to exert ourselves for more data.

The documentation afforded us a comprehensive overview of the process itself as a whole and its interfaces to the rest of the product process. It also gave us some insight with regard to instantiating the process model to a real software project and thereby clarified the management use of a process model in this particular process.

Furthermore, maybe most vitally, the documentation study was of avail in enlightening the context of the software process, which assisted us in deciding how to proceed in our modelling case, pertaining to the modelling requirements discussed above (see Subsection 6.4.1). As we learned of the process model already available and obtained a granular comprehension of the status of the actors' process knowledge, a general-level understanding of the model requirements could also be formed.

The documents also included a substantial amount of information regarding the current project. This process data was useful, despite the granular level of detail. For example, the project plan gave us directions to search for more detailed process data by informing us of the project organisation and task allocation. The direct effects of process documentation study are mainly limited to interviews. Documentation was used both in the preparation of interviews and in cross-checking the resultant data.

6.4.4 Participant observation

To complement the contextual information gained by studying process documents, we conducted some observation. Methodically, our observation is denoted as participant observation, because we participated in the software process as if we had been one of the "pure" process actors. In research literature, such participation most often relates to ethnography (see Schwarzman, 1993).

Our participant observation comprised three kinds of learning. First, by observing the actions of the process actors, we could indirectly discern the process they were performing. Second, it added to our conception of the context we were dealing with. Third, personally carrying out a portion of the software process enabled us to make direct observations of the course of the process. Any observations deemed essential were recorded in our case study database.

Located in a central place in a hall divided into small boxes with screens, we were able to spot any interesting situation taking place and attend the scene to obtain further information. In the beginning the observations made were quite arbitrary: we recorded into our data base anything perceived as important, mostly peculiar process details.

Much of the process data collected are perhaps best characterised as implicit, informal and contextual. We simply familiarised ourselves with the process we were going to model and obtained a general "gut feeling" of the process actors and their doings. This was valid the other way around as well, i.e., the actors familiarised themselves with us and our doings. We got acquainted with the characters of the actors, learned how active and spontaneous they are, what are their personal interests, their attitude towards novel things etc. This assisted us in directing our effort towards the target group, i.e. in paying attention to the intrinsic features of the target group while implementing the modelling process.

Enlarging the scope from process actors to the implicit and informal particulars of the process, we made observations about the nature of work, working atmosphere, communication and the actors' roles in the process.

In the course of our study, we also conducted some of the actual software engineering work carried out by the software process actors. Originally, our task was to specify, implement and test a software module to be released in the future as a supplement to the software being produced by the major project. We were supposed to exploit the software products already accomplished by the major project. The purpose of all this was to follow in the footsteps of the "pure" process actors thereby learning about the course of the process.

Deviating substantially from the original plans, we ended up merely specifying and designing our module. Initially, it was our intention to carry out as much of the actual software process as possible, but eventually it was discovered that an implementation of a minor part of the software module would not be even nearly as useful as a well done specification for the entire module. The focus of our work thus changed remarkably.

The other reason contributing to the deviation from original plans was sheer inexperience on our part. We underestimated both the duration of the modelling case and the duration of the software engineering work.

In retrospect, we should have allowed at least six months to the software construction, at least for the module we specified. Mere specification and design did not give much of a picture of the actual process. On the other hand, spending a half a year doing what everyone else does would be waste of resources from the process modelling point of view. Although there is little doubt that much could be discovered, it is at least arguable that only a fraction of the information obtained would be useful in modelling. Most of the data would probably relate to the software construct itself. It is therefore concluded that the chief deficiency of software construction as a method for process modelling data collection is its inefficiency. Only in constructing a detailed software process model would such data be beneficiary.

Albeit the software module construction itself was a failure as a data collection method, the specification of a real software module facilitated our communication with the actual process actors about process details. The usefulness of this experience in the communication is, however, impossible to measure.

6.4.5 Interviews

As in so many modelling cases and methods presented in the literature, our primary data collection method was interviews. The manner in which they were implemented may, however, differ from most of those found in past research.

The following subsections describe our interviews. First, their objective is stated (Subsection 6.4.5.1). Next, their course is presented (Subsection 6.4.5.2). Then follows a brief discussion of the interviewees (Subsection 6.4.5.3). After that, a more detailed discussion concerning the interview technique is presented (Subsection 6.4.5.4). Finally, the experiences regarding interview unpacking are reported (Subsection 6.4.5.5).

6.4.5.1 Objective

The interviews had a dual objective. First, there was the traditional goal of obtaining process data: who does what, when, how and why. Second, the interviews were to find out the actors´ requirements for the model and their interests.

6.4.5.2 Overview of Interviews

To begin with, we selected semi-structured interviews as the interview technique. Subsequently, we tested the technique by interviewing each other. After that, we conducted a few interviews to further evolve the technique. These four "pilot" interviews were carried out in a tandem style: one of us was an active interviewer and the other an assistant recorder. Gradually our interview technique matured, causing deviations from one interview to another. Although it was only during the first four interviews that the emphasis was on the technique and tools, no two interviews of them all were quite alike.

Having experience of the four pilot interviews and the subsequent pilot modelling (see Section 6.2) we could plan and schedule the rest of them. The remainder of the interviews, apart from one, were conducted in seven days. Each of the sessions lasted from one to two hours.

6.4.5.3 Role Division of Interviewees

We interviewed the entire software group whose process we were to model. To record the different types of information that persons occupying different positions in the process possessed, the interviewees were divided into three role-groups based on the pilot interviews and task allocation information acquired in the process documentation study.

Then, the questions were accommodated to the expertise of each the three groups as follows. Firstly, there was the software designers' group covering most of the interviewees. They were asked questions relating to their present, current and future tasks. For each task, such questions as what, how and why were asked (see 6.4.5.4.1 Interview Themes). Next, the team leaders (software designers with the special responsibility of team or subgroup leadership) were asked the same questions as the ordinary designers supplemented with questions about team leading activities and involvement in process development and project planning. Lastly, the project and group leader's questions concentrated mainly on project planning, controlling and monitoring.

Apart from the dissimilarities in the questions asked, the classification of the interviewees in accordance with their roles turned out to have little effect on the interviews. On the unpacking of the interviews it had no effect at all.

6.4.5.4 Semi-Structured Interview Technique

The semi-structured interview technique was used so as to facilitate rich answers. Therefore, the interviews were by far similar to conversations, where the process actors narrated their past and present task field as well as presented guesses of what the future might bring, and we refined the recital with questions. This conversation was guided by themes presented in the following subsection.

6.4.5.4.1 Interview Themes

Themes from literature. The themes used in the interviews were based mainly on literature. We used the commonly accepted list of questions a process model is expected to answer (see for example Curtis et al., 1992, p. 77): What has been done, Who has done it, when was it done, How and Why was it done. The Where-theme was intentionally left out, due to the irrelevance of physical location in our modelling case. Moreover, questions about application domain knowledge, customer requirements as well as communication and co-ordination were used (Curtis et al., 1988). All of these were seen as essential based on our observations as well.

Additional themes. In addition to the themes obtained directly from process modelling literature, we used some additional ones deemed as important based on process documentation, observations and the first interviews. These themes specified the how-theme and included issues which we wanted all the interviews to cover, such as tools, co-operation, planning, training, roles and some role-specific subjects as well as some factual matters such as the interviewees' background experience.

Model requirements. After the very first interviews we understood that registering implicit information was not enough to yield model requirements. Thus, a direct question regarding the interviewee's opinion on process details to be depicted in the model was added to the themes already mentioned. It was noted, however, that without any preparation most of the interviewees gave only general, vague and uncertain answers. We also had to rely considerably on specifying questions, which were leading, i.e. the questions in themselves constrained the answers. Hence, we used an e-mail query after the interview to supplement this question and to validate the data gained. The query is discussed below.

Use of themes. In the interview itself, the themes were normally followed by discretely asking the interviewee to specify something the interviewee had said, to cover the theme. This delicacy slightly suffered in the very first interviews as the interviewees did not always comprehend our remarks and comments, but soon we learned to phrase them adequately and to avoid misunderstandings. These confusions affected our theme list and also the interview technique. We found that the simplest way to speak the same language with the interviewees was to use their terms. In practice, the role of the interviewer was thus to keep the process actor talking.

The themes and the refining questions can be found in Appendix B. This theme list was used in an interview of an English-speaking process actor. As we never clearly documented the theme list, neither is the one presented in the appendix more than an outline paper used in one interview.

6.4.5.4.2 Course of an Interview

Scene. A typical interview took place in a meeting room with no external disturbance. The scene comprised three persons. First of all, there was the interviewer with a list of themes for discussion. His chore was primarily to keep the conversation going and to keep it from side tracks. Moreover, there was the assistant tending a laptop PC with audio recording software. He also followed the theme list to ensure that all essential themes would be discussed. If need be, he participated in the conversation to cover something the interviewer had missed. The third person present was naturally the process actor sharing his or her experiences with us.

Interview process. Basically, the structure of the interview was as follows: the interviewer lead, following the What-theme, the interviewee to narrating his or her chores in chronological order starting from the beginning of the current project or the point the interviewee had entered it. When the interviewee seemed to suppose that he or she had said everything essential but we were still missing something, additional questions were asked. These comprised the themes Who, When, How and Why as well the additional ones mentioned above (see subsection 6.4.5.4.1). Moreover, more specific What-questions could be asked to lead the interviewee into a finer level of detail in the currently discussed task or to lead him or her to the next task in chronological order. This cycle continued until the interviewee's present task was reached. At the end, the process actor's anticipation of what the future might bring was asked as well as some general, non-task-related questions, if some themes had not been properly covered earlier. Finally, we inquired the interviewee's opinion on what the future process model should depict.

Different interviews. It should be clear from above that no two semi-structured interviews can be identical. In addition to the variation owing to the significant role of the interviewee, our inexperience influenced to the same direction as well. Hence, all the interviews had their own particularities. Although the most substantial changes in the technique and questions occurred between and after the first four interviews, minor modifications were made throughout all the interviews.

Data obtained. By using semi-structured interviews we obtained answers with abundant details, formal and informal. As the interviewee could lead the conversation in directions he or she was interested in, the personal interests and attitudes of the process actor were unveiled. By registering the subjects that were most intensively discussed and the actors' attitudes towards the subjects, we gained some data regarding the model requirements.

6.4.5.4.3 Compromising Between Structured and Semi-Structured Interview Technique

Challenge of semi-structured interview. The semi-structured interview technique turned out to be rather challenging in methodological terms. On the one hand, the interviews should cover more or less the same issues yielding roughly similar information (i.e. process data). On the other, the technique allows the interviewee a more active role leading the conversation than structured interview technique does. In our modelling case this contradiction was amplified by our goal of recording the process actors' attitudes and opinions (i.e. model requirements) which they might not openly express.

From semi-structured to structured and back. As can be seen from the use of refining questions described above, we reformed our technique to a more structured direction. After four interviews we realised that we had gone too far, and returned to the more relaxed, free-form way of carrying out the interviews. For the remainder of the interviews we used the questions we had developed so far, but asked them in a more flexible order. That way the interviewee could feel he or she was steering the interview and we merely specified the details we were interested in. This emphasised the role of the assistant interviewer in keeping track of issue coverage. Still, we found it difficult to compromise between loose and rigid structure.

Experience as threat. Experience posed its own threat to the semi-structuredness; it was utterly cumbersome to keep from settling into a rut and follow the implicit hints of the interviewee. Here, however, the tandem technique was of assistance. We could alternate the interviewer and assistant roles between interviews, which gave the interviewer a chance to "wipe" the last interview off his mind.

6.4.5.4.4 Summary of Experiences with Semi-Structured Interview Technique

All in all, we found the semi-structured interview technique to be most suitable for our information needs. Carrying out the interviews in this manner gave us a strong picture of what are the essential process deficiencies that the model might remedy. Furthermore, it enlightened us about the "commoner's" perspective of a process model. This partly informal information functioned as one tool aiding us in the construction of a model understandable to those it is supposed to serve.

6.4.5.5 Unpacking Interviews

In a semi-structured interview, as in ordinary conversation, any comment in the interviewee's sayings may contain information relating to any one or more process details or not relating to the process at all. Therefore, classifying the comments proved to be impossible. After a short period of trial and error, we wrote reports of the interviews containing almost everything that could be heard on the audio record file; both the interviewer's comments and the interviewee's. From these "raw" reports, more specific "topic" reports containing solely model requirements, process data, or any other information could be more easily extracted. A sample of a "raw" interview report can be found in Appendix C and a sample of a "topic" interview report is located in Appendix E.

6.4.6 E-mail Query

To give the process actors another chance to answer our interview question about the model requirements (the desired content and uses of the model), we sent an e-mail query. There are two main reasons for using e-mail. First, it was an everyday tool in the software group, which meant that our message would be read. Second, we anticipated that expressing oneself in writing and without any haste would be a more convenient way for obtaining well-thought-out data.

6.4.6.1 Objective

The main function of the query was to validate the data collected in the interviews. Most of the interviewees had given some kind of an answer to the model requirement question. Most of these answers were, however, partial and even contradictory. This can be considered the dark side of semi-structured interviews: it is hard to get homogenous and comparable data. The query was to give all the interviewees the chance to comment on all the model requirements that had arisen in all the interviews so as to validate the answers. At the same time we were hoping to obtain additional information.

6.4.6.2 Questions

We assembled the mail as follows. To begin with, we listed and summarised the model requirements extracted from the interviews. Then, we marked the number of interviewees in the interviews of whom the requirement had arisen and who had taken a positive attitude towards it. Finally, we wrote a brief introduction containing a statement of the purpose of the query and an attempt to motivate the receiver. This list of requirements for the model with weightings for each requirement and an introduction was subsequently sent to all interviewees (see Appendix D).

6.4.6.3 Query Replies and Experiences

The query gave all the process actors the possibility to comment on all the requirements arisen in any interview. As more than half of the interviewees took advantage of this possibility, most of the requirements we had picked from the interviews were confirmed. The only exception was the communication theme, which we had thought to be important. Although most of the query respondents agreed upon its importance, the theme was perceived as irrelevant to model.

Although the e-mail query was concluded to be a success, there is one lesson that we learned. Notwithstanding that we, in the query introduction, had asked the recipients to comment on all the requirements, one receiver seemed to have been in the understanding that the requirements given in the query were exclusive. It could be seen in his response that he had tried to pick only one requirement. Nevertheless, this confusion had not harmed his answer from our viewpoint, because he had pondered the value of the other requirements as well. We obviously could have offered more detailed information about the use of the requirements in the modelling.

6.4.7 Data Collection Results

Albeit three data collection methods were used, interviews (including the e-mail query) were clearly the most significant one. Process documentation study and observation added primarily to our understanding of the context. They provided the initial information to guide the interviews and served as consistency-checking mediums for the interview results. Essentially, the data collection resulted in model requirements and process data, which we documented for later use.

The model requirements presented here were perhaps the most important in our consideration vis-à-vis the process aspects to be modelled and the level of detail of the resultant model. The requirements also specify the goals of the modelling, i.e., what is expected from the model and the modelling process itself. The model requirements are divided into two categories, each of which is presented below. The first category comprises general model requirements and the second more specifically the process details to be depicted in the model.

6.4.7.1 General Model Requirements

Compatibility. The process model should be matched with the higher level (product process) process description and it should be more specific than the existing process model. The new model could use the existing process description.

Generity. The model should be generic and hide project-specific details. On the other hand, the model should take into account the requirements of the development environment and avoid excessive generity.

Model uses. The model should help the project manager to plan and track projects more accurately. As for the software designers, it should describe how the software is produced. At the same time, it might facilitate communication with other processes.

6.4.7.2 Process Data to Be Depicted in the Model

The process model should present activities with inputs, outputs, entry and exit criteria as well as activity overlapping and backtracking. As for the inputs and outputs, it should be shown which phase-products are inspected or reviewed and when. As for resources, the role of the CASE tools used should be visible. The import of customer requirements should also be described, as well as their impact on the process, i.e., how they are taken into account in the various process phases. Furthermore, process interfaces should be depicted, i.e., what kind of information is shared, how and when.

6.4.7.3 About Requirements

The process actors had no wish to have resources (apart from some significant tools and external interest groups) to be depicted in the model, which was a surprise to us. In literature, actor roles or agents as resources are generally agreed to be important process information to be modelled (Curtis et al., 1992; Dutton, 1993; Yu and Mylopoulos, 1994). When asked about modelling resources, the process actors clarified their opinion by referring to the relatively small size of their process: there are only few separable roles, and the tools to be used are in most cases obvious.

Resource modelling is perhaps one example of the requirements being on a general level of detail. This owes primarily to our maidenly context: in the absence of a practical model of the process, it was the coarse, not the detailed level the actors were first interested in. This is a natural, logical way of approaching an unknown subject: first find out what it is about, only then more specifically how it works.

Another explanation for the omission of resources given by the process actors was the resulting difficulty of and continuous need for updating: adding resources to the model would reduce its generity. This argument can be traced to the small size of the process as well: the process is flexible and may change rapidly.

It is imaginable that many of the resultant model requirements are applicable to any software process model, provided that the goals of modelling are similar to ours and the software process has previously only been modelled on a coarse-grained level of detail.

6.4.7.4 Specification of the Model

Based on the requirements described in the previous subsections, a model specification document report was written so as to freeze the model requirements. The written specification functioned as a basis for the project leader and team leaders to commit to. The requirements and the specification reviewed and accepted by our management with only small specifications to the specification regarding compatibility and generity. It was also expressed that a textual description would be in place to describe the model in a lower level of detail. Subsequently, the specification was used to determine the process data we were to extract from the interview reports.

Most of the requirements were accepted with only some focusing. Because of the general nature of many of the requirements, more specific definitions had to be given in the specification document. The paragraphs below list those model specifications that were markedly refined from the corresponding requirements.

Software process and product process matching would be done, as before in the pilot model, through product process model milestones.

The model would offer support for project planning and tracking provided that adequate knowledge of the process is available. That is, the model itself, as we would create it, would not be enough to tell all a stranger needs to know about the process to be able to plan and track a project.

Owing to our limited information source, only a part of a single project, the model could only be generic to a certain degree. The model would be presented mainly on two abstraction levels, which would guarantee a sufficient level of detail but restrain the model from being too detailed. The bounds of existing process information would set the level of comprehensiveness as well.

Resources would be shown only if they add essential information to the model. The most significant external interest groups would be shown as well as their relations to various activities. Furthermore, the content and direction of the information flow in the interfaces would also be indicated.

Equipped with the model specification and the process data report, we could create the initial version of the software process model. This is the subject of the next section.

6.5 Initial Modelling

In this section, we present the experiences gained during initial modelling, which started after the data collection. Although considerable attention was devoted to the development of our PML at that time, this intricate issue is discussed in more depth in a separate section below (6.6), as it intertwines with other experiences gained over the same subject during the pilot modelling and the participative modelling.

6.5.1 Objective

Resting on the model requirements and specification as well as process data obtained, the main objective of the initial modelling was to construct a preliminary software process model to base the participative modelling on. Thus, the preliminary model needed to be readily understandable by the process actors.

6.5.2 Creating the Initial Software Process Model

The actual modelling, apart form the pilot study, started only after the most intensive data collection. In this phase, we constructed an initial process model to base the participative modelling on. Some of the pilot model diagrams were also utilised, although conformed to the new requirements.

Two steps can be perceived in the creation of the initial process model. First, the process model was created using a computerised drawing tool. This was a straightforward, routine task comprising reading and supplementing the process data by interrogating the actors and creating the model. This interrogation was profitable due to the process progress creating new process data. In parallel with the model creation, we wrote a design guide for our own use to ensure the syntactic and semantic consistency of the model. We also analysed the concepts used in the model so as to enhance the understandability of the process model by using logical concepts that would be as analogous with the process concepts that the actors use in their everyday work as possible.

Second, having finished the computerised version of the process model, it was verified against the model specification document produced earlier and a wall diagram representation was constructed.

6.5.3 Resultant Software Process Model

The hierarchy of the process model diagrams is shown in the picture below. Some of the diagrams (grey boxes) described process areas which were omitted from the official process model and were therefore previously unfamiliar to most of the actors. These process areas represent process support activities, which had formerly been left implicit.

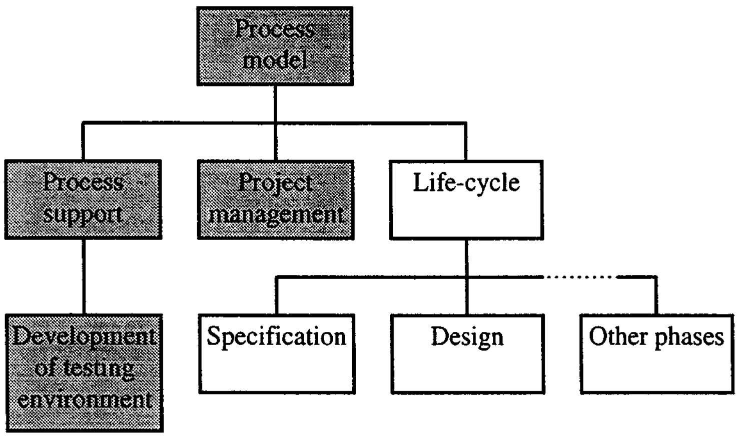


Figure 9. Diagrams Produced in Initial Modelling

The highest diagram was simply a composite diagram to illustrate the relationships of the three lower level activities. Process support covers such activities as process assessment and tool support. Project management, again, comprises the project planning and tracking activities executed primarily by the project leader. Finally, the life-cycle diagram represented the process as a modified waterfall model, whose phases were specified into a lower level of detail. The waterfall model was intended to replace the former official process model. In the relationships between levels of abstraction we had taken into account representational aspects. For example, the life-cycle diagram was simple to facilitate illustration purposes and contained mainly generalised composites of the phase products of the lower level diagrams.

6.5.4 Verification of the Initial Model

Having completed the process model, we verified it against the specifications produced in the data collection phase (see 6.4.7.4). As a result, it was concluded that the model fulfilled its specifications with minor exceptions regarding the representation of process interfaces: it does not specify which information flows to or from which external interest group. This information would be, however, included in the textual description accompanying the graphical model, as our management had wished in the model specification review. Later in the modelling case, we designed a template for constructing a project specific textual description of the software process (Appendix J).

Another specification regarding the process interfaces that was not fully satisfied, related to the customer requirements. For the most part, changes in customer requirements were indicated to cause backtrack to previously finished process phases. On the lower level of abstraction, they were depicted as inputs to activities along the process. The model did not specify how the requirements were taken into account in the process.

Obviously, many of the model properties, especially the types of data represented, are actually features of the PML used. Therefore, verification of the initial model boils down to verification of the PML. From above, it can be concluded that the PML had developed to fulfil the needs of this particular modelling case. What was needed to bring it to that state, is the subject of the next section.

6.6 PML Selection and Adaptation

The purpose of this section is to present our experience in applying the principle of PML engineering in practice. The issue was first seized before the data collection started and it remained current almost until the participative modelling. Thus, the step was actually executed in parallel with all the previous steps of the modelling process, and the experiences presented here are intertwined with those discussed earlier.

The structure of this section is as follows. First, phases of our PML engineering are walked through to give an overview of the events and their relations to the rest of the modelling case (Subsection 6.6.1). Second, the changes are more closely examined from the point of view of the process modelling perspectives (Subsection 6.6.2). Third, the advantages, disadvantages and possible uses of a multi-perspective PML are discussed (Subsection 6.6.3). Fourth, the most essential factors affecting our PML modifications are reported (Subsection 6.6.4). Fifth and finally, the course of discussion arrives at the single explicit tool we employed in the PML engineering: the conceptual model (Subsection 6.6.5).

6.6.1 PML Engineering Process

Survey. We began to survey the existing PMLs in parallel with the data collection step (see Section 6.2). As our knowledge of the process context increased and our plans for the use of the PML were specified, we were able to establish some selection criteria, including visuality, understandability, tool support and needed perspectives. One criterion particularly significant was the requirement that the PML would also have to be suitable for participative modelling (see 5.3 The Principle of PML Engineering and 5.4 the Principle of Participation).

Selection. Initially, VPML (Visual Process Modelling Language; Dutton, 1994) was selected without any explicit analysis or comparison procedure. Its process model was simple and illustrative and allowed multiple levels of abstraction, parallelism and iteration. It also combined multiple perspectives, which was another contributory feature. To better accommodate the process model notation to our modelling situation, we modified the PML. This modification process can be divided into three distinct phases.

First modification phase. Based on the first few interviews, the "pilot interviews", we simplified the VPML process model language by removing redundant properties and, on the other hand, added properties deemed useful in depicting the software process under scrutiny. We also constructed an OPRR (Object Property Relatioship Role) (see, for example, Smolander, 1991) conceptual model to clarify ourselves the concepts of the VPML process model language and their relationships (Appendix F).

As communication and information sharing among the process actors seemed to have a significant role in the process, we extended the VPML process model language by adding a notation for it. This notation was based on the actor dependency model (Yu and Mylopoulos, 1994) and linked to the process model itself. This notation described roles and the subjects they communicated over. We also added separate symbols for interest groups external to the process as well as a symbol for meetings and training. On the other hand, some VPML symbols were removed as redundant.

The first modification phase of the PML continued in parallel with constructing the initial version of the pilot software process model. This preliminary version of our PML was used in the participative phase of the pilot modelling. The modelling experience confirmed that VPML had provided a suitable basis for our PML, but some further conforming was needed.

Second modification phase. After the pilot modelling experience and the interviews we were in the position to further elaborate the PML to conform it to the model requirements and the specification derived from them. Since describing communication in the model was not deemed relevant in the model requirements, we removed the separate communication notation and combined its essential parts to the main model. Furthermore, from then on resources were to be included only optionally, i.e., when they would contribute significant information to the model.

Some additions were also made. These included the VPML temporal relations between activities that had previously been removed as well as a backtrack relationship to make explicit the returning to formerly finished activities.

Finally, we updated the pilot process model with the PML changes. Subsequently, the model was commented by the process support engineer. He considered the model understandable, but overloaded with different symbols and suggested that we should limit the number of different symbols to clarify the diagrams. However, we were only able to find one immediate opportunity to remove a symbol: the semantics of VPML's document symbol was extended to cover the semantics folder symbol as well.

Another observation made by our management was the ambiguity of the graphical process description. This could hamper the further development of the model and the process. It was expressed that a textual description would be in place to describe the model in a lower level of detail. Thus, we later designed a template for textually narrating the graphical description (Appendix J).

Third modification phase. After the participant observation (see Subsection 6.4.4), at the start of the initial modelling, some final changes to the PML were made. This third modification phase engaged building another conceptual model to conform the PML concepts with the notions used in the process context. At the same time, we managed to exclude some symbols. To ensure the consistency of the model, we recorded the modelling conventions in a free-form design guide. The guide contained information regarding usage and placement of symbols, terminology and representation of various process details. The PML symbols, their relations and meaning, again, were narrated in a symbol guide, which was primarily intended to enhance the model understandability to the process actors (Appendix G)

6.6.2 PML Perspectives Evolution

As reported above, the starting point for our PML engineering was the Visual Process Modelling Language VPML (Dutton, 1994). It is developed in the ProSLCSE project and is especially directed to modelling software processes. The PML consists of three models: infrastructure, information, and process model. The infrastructure model describes resources, such as humans and machines, i.e., the organisational perspective (see Section 3.5 Process Modelling Perspectives). The information model, again, describes the types of objects, such as documents that are manipulated by resource types, i.e., the informational perspective.

Finally, the process model language, which we selected as the basis for our PML, describes activities, temporal relations between them, the resources required to perform them, the objects required and produced by each activity, and the decomposition of activities to lower level activities. The process model language is based on the Data Flow Diagram technique and thus emphasises the functional perspective over others, but also the organisational, informational and behavioural perspectives are represented.

Together, the different models constitute an enterprise model. The three-model-structure is intended for proper representation of each perspective. Although the process model covers all the perspectives, informational and infrastructure models refine the informational and organisational perspectives. However, we found this structure superfluous for our purposes and used only the process model. The higher level of detail offered by the informational and infrastructure models was deemed unnecessary.

The changes made to the original VPML process model language affected mainly the representation of the organisational perspective. There were three such changes. First, the communication model designed in the first modification phase relates perhaps strongest to the organisational perspective, as it depicted the communicating roles and the subjects of communication. It can not be compared to VPML's infrastructure model because of its high level of detail. This was one reason for removing the communication notation: instance level information is self-evident or easily inferable and laborious to update. Second, the symbols for external interest groups also representing the organisational perspective, on the other hand, maintained their position in the PML. Third, only three instead of VPML's ten (five for types, five for instances) resource symbols were used and in the second modification phase they were decided to include only optionally, which weakened the representation of the organisational perspective.

Another perspective affected by the changes made was the functional perspective. Namely, the number of different product symbols was reduced from five to three.

Should the reader wish to make a more accurate comparison of the original VPML and our PML derived from it, this is possible via VPML construct overview (see ProSLCSE

Project, 1997) and Appendix G containing the VPML symbol catalogue and our symbol guide, respectively.

6.6.3 About a Multi-Perspective PML

VPML's process model language and the resultant PML both combined more than one perspective in one process description. This has its advantages and disadvantages. The notation is easy to learn and intuitively understandable, which saves the actors' time. Thus, excessive use of resources can better be avoided at the beginning of process modelling. This may be useful in the initial phase, while trying to understand the current process, because at the beginning no valid decisions may have been be made about the continuing of the improvement.

This means the modelling is not granted the full and undivided attention of the process actors, but has to survive on the bits and fragments of attention paid to it and the personal interest the process actors have towards it. In this such a situation the decision to use a single notation that is easy to understand is thereby easily justifiable. This was exactly the case in our research.

The risk of a multi-perspective PML is in cluttered models overloaded with different symbols. Thus, none of the perspectives may be represented clearly. To avoid this, all the perspectives can not be represented in the level of detail that they could be if they were depicted in different models. This may be considered as a disadvantage. Partially because of this, our changes brought VPML to a simpler direction.

As for the future, it is conceivable, that once a single multi-perspective model of the process exists, multiple models with a single perspective each may be introduced. Conditions for doing this are sufficient resources and that the multi-perspective model does not expose the process to a sufficient level of detail or one or more process aspects are ill-represented. Then, the role of the multi-perspective model would be to act as a large scale map that gives you some idea of where you are.

6.6.4 Factors Affecting the PML Modifications

The two subsections above provided a closer look into the changes made to the PML. The causes of these modifications however, have yet to be discussed. The changes were heavily influenced by the process context, but also studying process modelling literature was found useful in the initial phase. Some factors affecting the selection of VPML and its modifications are discussed below.

Process modelling literature. Some initial selection criteria were found in process modelling literature. Both the PML classification attempts and discussions about desirable characteristics of a PML presented in Section 3.6 Software Process Modelling Languages provided us with a general conception of features to look for.

Model requirements. After we had selected VPML as the basis PML, the data collection phase, mainly the interviews, added to our knowledge of the process aspects considered important and some more general requirements for the model. These were collected to a model requirements document, based on which we wrote our model specification (See Subsection 6.4.7.4). These, in turn, had a direct impact on the PML in its second modification phase (see Subsection 6.6.1).

In addition to the model requirements obtained via interviews, some casual observations of the process actors directed our effort. For the most part these were encouraging, except for one. As reported above, the ambiguity of the graphical notation had been paid attention to by the process support engineer before the initial modelling. This appears to contradict with our "unformalising" of VPML's process model language. This engineer, however, occupied a leading position in the software process and was thereby more experienced than most of the actors. He also had a fairly good viewpoint to the process and thus already possessed an understanding of it. Still, our goal was to support the average process actors.

Nature of the process. The nature of the software process influenced the development of the PML to a considerable extent. Already at the beginning of the study we came to realise the hectic nature of the software process: the process actors would not have the time to learn any complex PML. Therefore, the PML would have to be simple and intuitively understandable. This stressed the visuality of the language as an initial selection criterion.

Besides the hectic nature of the software process, its virginity amplified the requirements imposed by actor participation (see Section 5.4 The Principle of Participation). It is also feasible that the level of abstraction developed to a more general direction and the model ended up being less formal than the original VPML because of the virginity of the process context. Conceivably, process actors are first likely to attempt to attain a general idea of the process and only then find use for a more detailed model. This is also the idea of facilitating understanding being the first goal, which provides the foundation for others (see Section 3.3 Software Process Modelling Objectives).

6.6.5 Conceptual Model

To better tackle the PML engineering we constructed a conceptual model depicting the concepts of the PML and the relationships between them. This was our primary tool in modifying the PML and therefore deserves some attention.

The first conceptual model (see Appendix F) was developed in the first modification phase. In the third modification phase, we understood that using the process concepts already familiar to the actors would be beneficial in participative modelling. Therefore, a second conceptual model was built, which integrated the process concepts with those used in the PML so far (see Figure 10 below). Compared to the first model, this second one contains some new concepts, renames others and also excludes a few. The models are rather sketches than formal models, but they were nonetheless sufficient for our purpose.

Although we did not apply any explicit SPM method engineering process, conceptual modelling was found useful in moulding the PML as a whole. Especially the adaptation of the PML to the concepts used in the software process was facilitated. Due to this attempt we also came to realise the hindrances ambiguous process concepts impose on the modelling. Many of the notions used in the process context referred to multiple real-world objects and vice-versa. With the help of process actors we uniformed the concepts and then matched those with the ones of our first conceptual model. It was concluded that the subject would require more effort than we could offer. This ambiguity may be attributed to the maidenly context.

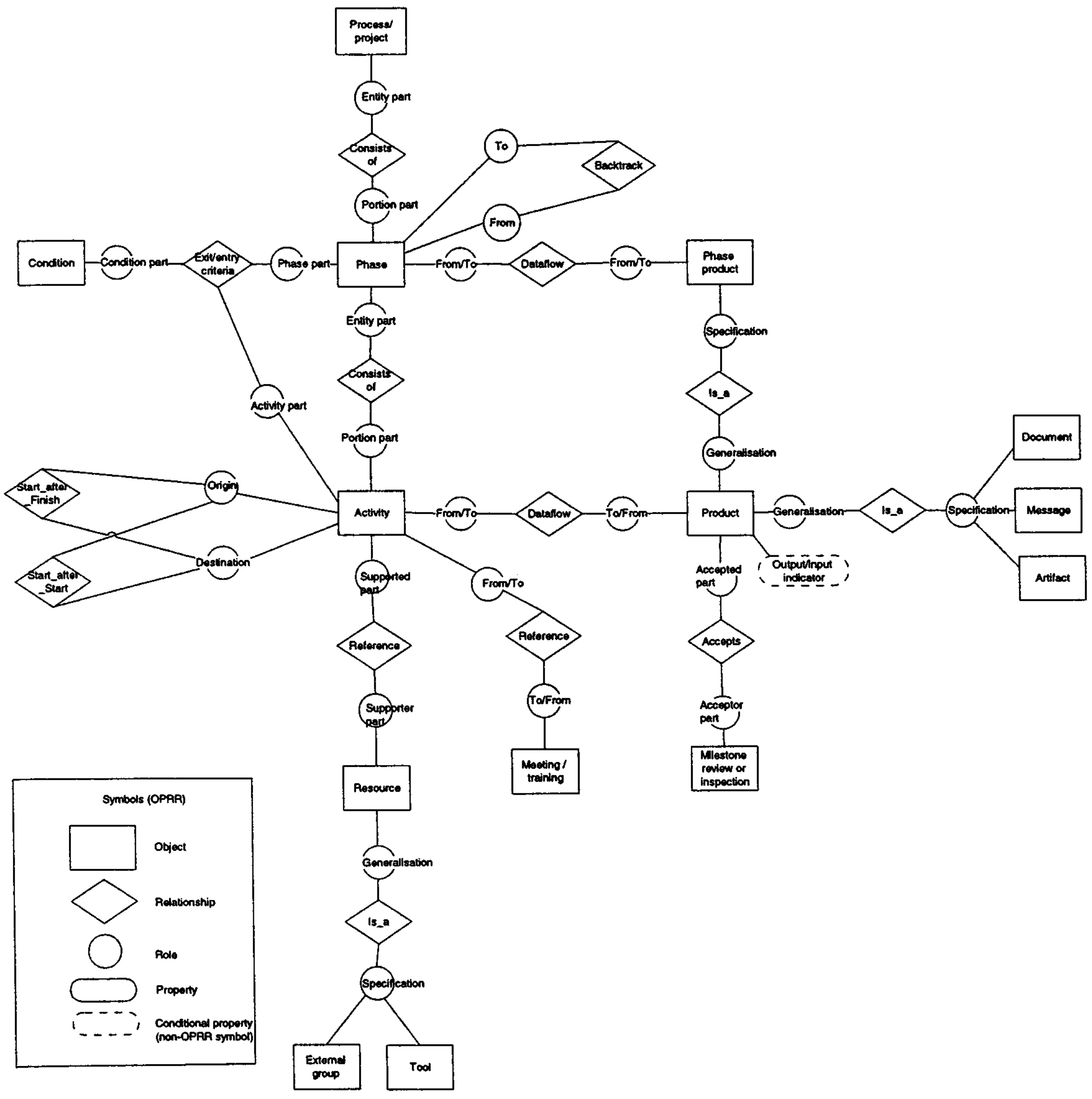


Figure 10. A Conceptual Model of the Final PML

Armed with the PML fitted to our needs and the initial software process model constructed and verified we headed for participative modelling, which was to crown our modelling case with the indispensable touch of the process actors. The next section presents our experiences in this pursuit.

6.7 Participative Modelling

This section exhibits the experiences we gained while executing the fourth and final step of our modelling process, the participative modelling. The following two subsections state the objective of the participative modelling (6.7.1) and provide an overview on the course of the step (6.7.2). The subsequent subsections examine the activities in more detail. First, in Subsection 6.7.3, the preparing for the participation is considered. Second, the commenting sessions are discussed (in Subsection 6.7.4) as well as two related issues, group division (6.7.5) and model development (6.7.6). Lastly, the final session concluding the modelling case is presented (Subsection 6.7.7).

6.7.1 Objective

The objective of the participative modelling was to involve the process actors in the modelling exercise to enhance their understanding of the software process as well as the correctness and completeness of the model. On the other hand, the actors were to validate the model, i.e., to check if it served its purpose and if not, then insist modifications.

6.7.2 Overview of Participative Modelling

Participative modelling was set about as soon as initial modelling had been finished. To begin with, we divided the process actors into groups consisting of two to four persons united by their work tasks. Next, we held an brief introductory meeting to present our plans for the participative modelling and the initial process model. Then it was time for the participation itself. The implementation of participative modelling had been designed based on pilot modelling experience. Thus, we arranged two rounds of modelling sessions which the actors attended in their groups. In these sessions, we went through the model diagrams and the actors might make corrections and additions to them as well as discuss presentational aspects, model generity etc. Finally, a process improvement brainstorming session was held. This was the last major exercise of our modelling case. Afterwards we sent an e-mail inquiry to collect modelling feedback from the process actors.

The following subsections narrate in more detail the progress of participative modelling and report the associated experiences.

6.7.3 Preparation

Motivation and process thinking background. Ever since the start of our study, the process actors had been kept aware of our goals, plans, working methods and progress. This was intended to motivate them to share their information for the sake of our data collection and to stimulate process thinking in the course of their work. From our experience during earlier phases of this study, such mental preparation to the spirit of our modelling had been either successful or redundant. Throughout the modelling case, the actors had been interested in our doings. During initial modelling some had come and discussed their process, methods and modelling with us, which was, in fact, participation in the very same sense that we use the word. Naturally, there was variation in the actors' enthusiasm towards modelling their software process. Some were actually reluctant to use their time for the modelling and pessimistic about the usefulness of its final outcome. Thus, there still was motivation work to be accomplished.

Before the commenting sessions, we therefore arranged a briefing to present the goal of participative modelling and the working method including the group division we were suggesting for the commenting sessions. The actors were delivered printouts of the initial process model covered with an introductory letter. The printout was intended to enable the actors to familiarise themselves with the model on their own before the commenting sessions. Moreover, a time schedule for the sessions was suggested.

The briefing was the very first occasion in which the process actors were given a paper copy of the model. Until then they only could have surveyed the wall diagrams in our box or the pilot model on the file server. Most of the actors leafed through the printouts, some searching for process areas familiar to them. The reaction showed interest, which was a positive sign.

6.7.4 Commenting Sessions

The aim of the first commenting session round was to familiarise the process actors with the entire process model. On the second round we steered the discussion towards analysing the model mostly in regard to its correctness, completeness and generity, but also consistency.

Basic scene. The commenting sessions took place in our room, where the wall diagrams were. In the sessions one of us presented the diagrams to the group in turn, and the other took notes and made corrections to the model as they arose in the course of the conversation. The model was presented on an activity by activity and product by product basis.

Rounds. On the first commenting session round all the diagrams were went through. This was intended as a introductory round for all the actors to familiarise themselves with the entire process. Subsequently, the wall diagrams were updated with the changes and corrections suggested in the first round. On the second round most of the diagrams were merely skimmed through to improve the participants' overall picture of the model and refresh it with the changes made. Only those diagrams of particular interest to the group in turn were more deeply entered into. This was found to be a good practice, since most of the discussion arose concerning the process areas familiar to the group.

Third session. In addition to the two model development rounds we offered to arrange one or more discussion oriented sessions, where process actors could ask each others more specifically about process areas unfamiliar to them. As this roused no interest, the idea was abandoned.

Round roles vs. plans. The planned roles of the two commenting session rounds did not quite match the realisation. The first round ended up being a commenting round in its own right although we had anticipated it to act only as introduction. The models had to be updated already after the comments of the first round to take into account the newly arisen information. The second round, on the other hand, resulted in relatively few changes. Thus, it can be concluded that the actors had little difficulty in reading the model, which was one of our original goals.

Comments and updates. The received comments concerned the entire model. The model was updated based on the comments. In the updates inputs and outputs of activities were specified; missing products were added and redundant ones removed. Temporal relations of activities were specified and some activities were merged into others. Some comments also influenced the representation of the external boundary of the process. The most penetrative observation in the commenting sessions was perhaps that everyone gave an opinion on at least one matter about the model and practically all parts of the diagrams were commented upon.

Use of the model. The participants themselves also made some valuable observations of their own process. One of the leaders came to realise, that the rush in the process had pruned away the test planning for a part of the software. One of the actors had previously been thinking the shift from specifications to implementation and was now able to use the model to point out the deficiencies he had observed in the process. The division of the process into phases as a whole aroused a lot of discussion among the participants.

All in all, the commenting sessions had succeeded in involving the actors in the modelling of their process. This is consistent with the feedback we received afterwards. Referring to the discussion about motivation above, it is concluded that in the end all the actors had high esteem towards our effort and were willing to help in any way they could.

6.7.5 Group Division

The actors attended the commenting sessions in groups of two to four persons according to their expertise and work tasks. We attempted to collect people with similar viewpoints of the process into each group to enhance conversation.

The group division turned out to be neither a significant success nor a total failure. Most of the groups discussed on process areas of common interest, but some groups were – in spite of homogeneity of tasks – quite heterogeneous in interests. It could be that a loquacious character dominated the sessions while the others were reticent. However, here we enter the problem field of group dynamics, in which neither of us is an expert, and thus, leave further discussion to those who are.

The process actors in a leading role formed one group, which was a successful solution. An interesting observation about the leader group compared to other groups is that they sat by far the longest sessions with us on the first commenting round, but had little to comment on the second round of commenting sessions. They were obviously quicker to grasp the idea of the model and to make their own conclusions than the others. They also could see the significance of the process model and its possible uses more clearly. Hence, they commented on the generity of the model already on the first round.

6.7.6 Model Properties Development in Commenting Sessions

Participative modelling improved the model in four major aspects. These are: correctness, completeness, consistency and model generity. Each of them are discussed in a separate paragraph below.

Correctness. We consider many of the changes resulting from the commenting sessions as enhancing the correctness of the model. Until the participative modelling, the model was a product of our own interpretations, and after the participative modelling it can be seen as a representation of the designers' views as well. We might have, erroneously, modelled some action as two distinct activities or pictured too strict temporal relations between activities. Based on the comments, some activities were merged into each other and iteration between activities was added. Of course, there are no guarantees that the model was totally correct after the participative modelling. We can merely say, that the correctness was significantly improved.

Completeness. In adding missing items to the model the completeness of the model was enhanced. We obviously had not been able to collect all the relevant process data.

Consistency. The terminology of the model became more consistent. The inconsistencies were due partly to the unsettled terminology of the process and partly to our own misperceptions. We were not always able deduce the meanings of all terms and couple them with the correct subject matter. As a result of the commenting sessions, also the representational consistency was amended. When presenting the models, we ourselves noticed inconsistencies in picturing the same matter in different parts of the model. Some process actors also made similar observations.

Model generity. On the second round we raised a question about the model generity. The leader group had already discussed it spontaneously on the first round. Generity is an important issue especially on the life-cycle level providing an overview to the entire process. Generalising the model entailed removing activities and products which are most likely only related to the current project or would otherwise only be needed once. Conforming terms from instance to type level was also a part of the generalisation exercise. Although generity issues could be taken into account in multiple places, it proved to be a too difficult task to generalise the entire process model based on the experiences of a single project which had not yet even reached its final activities.

6.7.7 Final Session

Objective. To pack the participative modelling experience, a process improvement brainstorming session was arranged. Its primary objective was to list the most essential weaknesses of the software development process under study. The deficiencies needed not relate to the process model: any improvement requirement would be accepted. If the actors had some implementable improvement idea on their mind, we would record them as well. This was the secondary objective.

E-mail. To start with, we sent an e-mail inviting the actors to participate in the final session and a sauna-evening afterwards to the actors. The sauna-evening was intended as an informal social occasion relaxing the atmosphere of the preceding final session. In the mail, we explained the purpose of the session and urged the recipients to prepare for the occasion by trying to discover five process weaknesses and possibly corresponding improvements. To inspire the contemplation, we added some deficiency observations and improvement suggestions to the e-mail. These ideas would then be used as a basis for brainstorming group work.

Brainstorming process. The session itself began with a presentation of the results of a previous CMM self-assessment to inspire the idea generation. Subsequently, the process actors were divided into groups, which dispersed around for about twenty minutes. During this time, they were supposed to come up with as many as five process deficiencies optionally some improvement ideas. Next, one member of each group presented the group's ideas, which were listed on one overhead transparency. Based on this list, the groups were then expected to arrive at three first rate items, each of which could be a specific reformulation of a previously presented problem, an entirely new weakness or something to improve the process. After another twenty minutes, the groups' products were again presented similarly to the first time. These ideas were subsequently summarised on one transparency, and briefly discussed.

Produced ideas and the role of the brainstorming. Clearly, the process actors adopted a serious attitude towards the matter and set themselves to produce usable ideas. We recorded a multitude of various ideas. To keep the participants' concentration in the subject, we had to time to further elaborate the recorded ideas or devise action plans. Nevertheless, many of the actors considered such an occasion profitable enough to be held more often. Its merit was in tearing the process actors off their work duties and stop them to ponder the occurred problems.

Based on our experiences from the final session and the comments received afterwards, the next section attempts to characterise the effect of the modelling on the process context.

6.8 Outcome of the Modelling Case

Measuring modelling success, i.e., the actors' improved understanding of their process, is extremely difficult. Of the diverse brainstorming ideas, it is impossible to deduce, which would not have been presented if not for our process modelling, i.e., which were the product of the modelling case and which were not. No measurement of the possibly improved process comprehension of the actors could thereby be made by analysing the ideas. The predominant opinion, which we agreed to, was that many of them had been known long before our modelling exercise. This applies mainly to those relating directly to the software engineering, such as testing coverage and fluctuating customer requirements.

The only metric unmistakably indicating improved understanding we had was an experience inquiry sent to all those having participated in our modelling case. In the responses, the process actors evaluated, among other things, the role of our work in relation to their conception of their process. They considered the modelling case useful in enhancing their understanding of the process, mostly by clarifying their notion of it.

In the course of our modelling case, a variety of experiences were collected, relating to each of the fundamental principles presented in the previous chapter as well as the modelling life-cycle used. These are summarised and discussed in the next chapter.

7 LESSONS LEARNED IN APPLYING THE PRINCIPLES IN PRACTICE

In this chapter, we recall the fundamental software process modelling principles presented in chapter 5. Their implementation in the modelling case of the previous chapter is contemplated and the experiences associated with the implementation are summarised and discussed. This supplements the fundamental principles with the lessons learned in their application.

The principles are intended for software process modelling, which is conducted in a maidenly context with the objective to facilitate understanding and communication of the software process. Within these presuppositions, they are meant to be context-free, constituting guidelines to be adapted in any modelling situation.

The following sections first summarise the modelling experience from the viewpoint of the principles. To begin with the three more practical principles, multiple data sources, PML engineering and participation are discussed in sections 7.1, 7.2 and 7.3, respectively. Then, summarising these three, the more abstract principle of flexibility is focused upon in Section 7.4. The first paragraph of each of these four sections recalls the principle from chapter 5. After the experience summary follows two sections discussing aspects arisen during the modelling. Section 7.5 focuses upon the application of the principles and Section 7.6 contemplates issues related to the modelling life-cycle.

7.1 Principle of Multiple Data Sources

Data collection for process modelling should be based on multiple sources of data.

Most software process modelling is based on data obtained from multiple sources. We emphasise the simultaneous use of the various sources to yield more detailed, accurate and convincing data.

This principle was applied mainly in the first step of the modelling process, data collection. The multiple sources used included process documentation study, participant observation, interviews and the subsequent e-mail query. In the modelling case, the three activities of the step applied these as follows.

1. Process context study. Process context information was collected via process documentation study and participant observation. This information guided our later actions.
2. Collecting model requirements data and process data. This main task of the data collection was tackled through all the multiple sources.
3. Elaboration of the collected data. In finalising and documenting the data collection results, an e-mail query supplementing the interviews was used.

Process documentation afforded us an initial overview of the software process. The documents also clarified the possible management use of a process model and enlightened the status of the actors' process knowledge. As most of the documents were written mainly for administrative (project planning etc.) purposes, however, they were on a general level of detail and played only a minor role in the every-day process. Thus, they primarily helped us in navigating to the essential process data.

Participant observation contributed implicit, informal and contextual information to the modelling. We familiarised ourselves personally with the process actors and their doings and vice versa. Thus we and the actors learned to "speak the same language". This was slightly facilitated by the software specification and design work carried out. The resultant contextual information was used to direct our later efforts.

Interviews were the most important data source. Using the semi-structured interview technique we obtained formal and informal process data and model requirements in the form of personal interests and attitudes of the process actors. The e-mail query validated the heterogeneous interview results by allowing all the actors to comment on all the model requirements derived from the interviews. In retrospect, it could have been wiser to first concentrate on model requirements and only then on process data. In our case, the requirements were clearly formulated only after the interviews and subsequently much of the interview data proved to be redundant.

Our experience suggests that the use of multiple data sources results in more accurate and comprehensive information. Process documentation provided a useful overview with some historical data. Together with participant observation yielding informal implicit information, it helped to bound the modelling to the process context. Thus, we were able to adapt the interviews for our target group, thereby exploiting the best the interviewees could offer. The structured questionnaire, again, validated and supplemented the interview results. Further additions were acquired via ad hoc inquiries during the initial modelling phase, which related more directly to the construction of the initial process model (see Subsection 6.5).

All the data sources had distinct, although overlapping, roles in our data collection. Had we used only one source, the information would have been severely harmed. For example, process documentation study excludes implicit information, and interviews would have required several iterations to pierce to a sufficient level of detail of the essential process features. It may be concluded that used together, the sources complemented each other forming a coherent whole, but none of them would have sufficed alone. Each of the sources were in place providing different kind of data.

After the data collection phase, we move on to the modelling itself. The next section presents our PML engineering experience in a nutshell.

7.2 Principle of PML Engineering

PML(s) should be selected and modified in accordance with the process context.

The modelling language used plays a significant role in software process modelling. To facilitate understanding and communication of the process, the PML should be simple and illustrative enough for the process actors to understand, but its syntax and semantics should also be sufficient for modelling the process.

The principle of PML engineering was utilised in the second step of the modelling process, PML selection and adaptation, which comprised the following tasks.

1. Survey on the existing PMLs. Process modelling literature was reviewed to find a PML that would be suitable for our modelling situation.
2. Selection of the most suitable PML(s). VPML was selected as the basis PML. The selection was conducted by ourselves in an ad hoc fashion, i.e., no explicit evaluation and selection method or framework was applied.
3. Modification of the selected PML(s) to fit into the modelling situation. The syntax, semantics and representational properties of the VPML process model notation were moulded to conform our process context. Conceptual modelling was used in the modification, but no explicit SPM method engineering process was applied.
4. Testing of the resultant PML(s). The resultant PML was prototyped in the pilot modelling case, which involved both developing and testing the language.

Albeit the selection of PML happened in an unmethodical manner, the chosen PML, VPML, proved to be suitable for our purposes. It was visual and intuitively understandable. VPML's three-model structure, however, was simplified due to its superfluously high level of detail, and only the process model was used as a basis for the PML engineering. Although the process model notation was supplemented with a separate model describing communication, this too was later removed as self-evident and laborious to update.

As a whole, the VPML evolved to a more simple form. Only some symbols were added. It was seen that this way the language served the hectic and maidenly process context better. Here, the model requirements had a significant role in setting a clear objective for the PML modification.

Further, as we came to realise the hindrances ambiguous concepts impose on the modelling, we used conceptual modelling to conform the PML concepts to those used in the process context. This conforming eased the actors' understanding of the model. Conceptual modelling was found useful in moulding the PML as a whole.

From our experience, it seems that by engineering the PML according to the process context the language becomes closely fitted to the needs and capabilities of the modelling situation. This issue is further discussed below, associated with the principle of flexibility. It must be noted, however, that our PML engineering experience provides only a limited view into the possibilities of the principle. In the absence of comparative experience, no strong practically founded assertions about the inherent possibilities of the principle may be made.

Besides the principle of PML engineering, the principle of participation, which is dealt with in the next section, is engaged with the model construction.

7.3 Principle of Participation

Process actors should participate in the modelling.

To enhance communication both between the actors and the modellers and among the process actors as well as to facilitate the actors' understanding of their process, the actors should participate in the modelling. Furthermore, to accommodate the actors' needs and wishes, the modelling process has to be flexible in every turn. Especially, the PML has to be selected and modified according to the process context.

The fourth step of the modelling process implemented the principle of participation. This participative modelling step consisted of four activities.

1. Preparation. We informed the process actors via e-mail and held a briefing, in which the initial software process model was handed to them.
2. Participative modelling. The process actors commented the models by attending commenting sessions in small groups. In and between these sessions the model was edited according to the comments.
3. Assessment of the depicted software process. A final session was arranged with the goal to expose the most essential weaknesses of the process.
4. Assessment of the modelling process. The process actors' comments regarding all the phases of the modelling process were collected via e-mail.

In our modelling case, the modelling had also management support and the actors were, apart from a few exceptions, interested in our doings. Thus, it was easy to keep them motivated by openly distributing information about the course and progress of the modelling and its effect on their work. Apart from one exception, no objections to participation arose.

The commenting sessions were also successful: the actors understood our PML quicker than was expected and all parts of the diagram were commented upon. This commenting rendered the models more correct, complete and consistent, even though no absolute measurements were made. There occurred also some assessment of the underlying software process. Most importantly, the final session after the commenting sessions indicated a raise in the level of the actors' process thinking. To conclude, participative modelling succeeded in involving the process actors and facilitating understanding and communication of the process. This was also expressed in the e-mail comments.

Now, having recalled and summarised the three more practical principles, the principle of flexibility summarises them in the following section.

7.4 Principle of Flexibility

Software process modelling should be adapted to the process context.

This principle emphasises the contextual relationships of the modelling and calls for context-orientation throughout the entire modelling process. This is considerably eases the enhancing of communication and understanding of the software process, because such a modelling goal deals with the process actors' awareness of the software process executed in it. The purpose of the modelling is to lead the process actors to recognising their process so as to be able to improve it.

The principle of flexibility is the most abstract of all the four principles presented here and has no direct practical implementation in the modelling life-cycle. Instead, flexibility is implemented through the other principles, PML engineering before anything else. On the other hand, flexibility is also required by the other principles, primarily by participation.

Actor participation necessitates flexible adaptation of the modelling according to the process context. Herewith the modelling can intensively exploit the capabilities of the context and accurately meet its objectives and requirements.

In order to adjust the modelling exercise in accordance with the process context, accurate information about the context has to be available. The provision of such information is enhanced via the principle of using multiple sources of data. This unveils the capabilities and needs of the process context. The primary tool of modelling to meet these needs and to exploit these capabilities is the modelling language. Therefore, the PML, above all, has to be flexibly modified to suit the context.

From our experience, taking account of the particularities of the process context renders possible the intense bi-directional communication between the process actors and the modellers. This improves the efficiency of the modelling, i.e., all of the modelling exercise is utilised as effectively as possible.

Next, the principle of flexibility leads us to discuss our findings in more depth.

7.5 About Principles in Practice – Discussion

One regal thought of the principles is to bound the modelling to the process context, so that when the model is developed, the modelling process itself increases the process actors' understanding of the software process. The key issue in bounding the modelling to the process context is flexibility. All the other principles include the implementation of this most abstract one. The most obvious example is, of course, PML engineering.

The three more practical principles, multiple data sources, PML engineering and participation all have their distinct functions in "contextualising" the modelling. Multiple data sources provide information about the context, PML engineering moulds the PML for use in the context and participation involves the process actors, the very agents of the context, to share their contextual knowledge and to increase their understanding and communication of the software process.

The following subsections are organised as follows. First, Subsection 7.5.1 ponders the application of flexibility in modelling. After that, Subsections 7.5.2 and 7.5.3 discuss issues related to participant observation and actor participation, respectively. Finally, Subsection 7.5.4 contemplates the effect of the maidenly modelling context on modelling work.

7.5.1 Flexibility Throughout the Modelling Process

Of the three more practical principles, actor participation provides a guideline for achieving the modelling objective. This is greatly facilitated by obeying the principle of flexibility throughout the modelling process. The actual degree of flexibility in any modelling process and the effect of applying the principle in ours are, however, impossible missions. Nevertheless, the advantages of flexibly accommodating the modelling to the process context are easily inferable. Let us consider these in the light of an imaginary fixed modelling process.

Flexibility in data collection. With a fixed variety of fixed data sources, which are not applied according to the modelling situation, the accumulation of a detailed body of process knowledge is likely to require more iteration than if the sources were selected and techniques adapted according to the context. For example, a planned comprehensive documentation study may be watered down by the poor level of process documentation or interview results may be impaired by poorly motivated interviewees. These shortcomings need to be compensated with other data sources that more closely fit the capabilities of the modelling situation.

Flexibility regarding the PML. Furthermore, meeting the needs of the process context with an unbending PML imposes enormous pressure on the selection of the language. Even if a usable one was found, which is of course possible, using any language without modifications is likely to exclude some process details that the model should contain and, on the other hand, depict process characteristics that are redundant in the model and thus hamper the presentation of the essentials of the process. Additionally, when the preconditions and requirements change, another language would have to be employed.

PML engineering is less costly and provides the modelling with a language with exactly the desired properties.

Flexibility in participation. Finally, if the imaginary inflexible modelling process managed to reach the participative modelling phase and to keep the process actors motivated, a premeditated form and intensity of participation may jeopardise the entire effort. It should be obvious that involving the actors has to happen on the actors' terms, not on the modellers'. Ideally, the actors' role in the modelling would be free to move between "informant", "reviewer", "participant" and even "modeller" as well as mix different roles. For example, the actors' process thinking may not be on a level sufficient for more than pondering specific questions directly related to their personal task fields. On the other hand, they could be capable of modelling their own process with some guidance from process modelling experts. In this case, they might be frustrated with a more controlled form of participation.

No universal recipe. Despite the obvious advantages of flexibility, much of the resulting benefit depends on how the principle is applied. Our experience merely suggests that every step of the modelling process should be elastic to adapt the effort to the modelling situation at hand. No universal recipe to tell how this should be done is offered.

Risk of flexibility. Bending according to the requirements of the process context has also its hazards. Without sufficient control and clear limits, the focus of the modelling may be blurred and the process will result in nothing usable or at least modelling resources are wasted. Similarly to software engineering, software process modelling has to balance between standardisation and flexibility.

To be able to flexibly accommodate the modelling process to the process context, the modellers have to possess sufficient knowledge of the modelling environment. The next subsection contemplates the technique we used for obtaining such knowledge, participant observation.

7.5.2 Contextual Information via Participant Observation

Profitability of participant observation. We found participant observation extremely profitable in data collection. Involving ourselves in the process to be modelled offered us a viewpoint inside the process. We managed to establish personal relationships with the process actors, which profited our later efforts. We could observe the available data sources and had unrestrained access to them. Most importantly, observation afforded us a profound body of contextual knowledge, the importance on which is infeasible to measure. Such information was utterly useful in pondering how the process actors' understanding of their process might be improved. Had we conducted our study by only occasionally visiting the site of the modelled process, this would not have been possible.

Limitations of participant observation. Participant observation may be criticised of the potential biases produced (see Yin, 1994). These may result from the investigator losing objectivity when he or she gives up the more traditional external position of an observer. The participant-observer may become a supporter of the group being studied or the participant-role may require much resources, so that no time is left for making observations. We are not in a position to either unreservedly accept or totally reject these criticisms. According to our experience, participant observation is useful in eliciting contextual information. Still, we perceive it possible that the use of this technique is limited by rather concrete and practical constraints. There are two such limitations.

First, participant observation in an industrial setting largely depends on the company whose software process is modelled and the relation of the modeller(s) to the company. It may be that any intense involvement in the process' internal actions is not wanted. This barrier might be overcome if the modeller(s) are employed by the company. Another conceivable limitation for applying our promising experience in other modelling cases is its resource consumption. Such detailed familiarisation with a software process requires a lengthy period of time, and the amount is positively related to the size of the process to be modelled.

Jumping from the beginning of our modelling process to the end, the next subsection focuses on the prospects of participation.

7.5.3 Advantages of Participation

Based on our experience, actor participation appears to profit the model quality in three ways. First, with the actors sharing their knowledge of the software process with the modellers, participation provides the modellers with an extra data source. After all, they are the experts of their own process (see Avison and Wood-Harper, 1990, p. 131), and each of them can validate and enhance the accuracy and completeness of the model as well as remove redundancies by providing information relating to his or her task field.

Second, the surfacing information might otherwise be inaccessible not only to the modellers, but possibly also to other process actors. In a discussion, however, this information is available to the rest of the participating actors as well. Thus, with this communication among the actors, one of the original modelling objectives is already partially achieved during the process modelling process: facilitating communication.

Third, similarly to process data, the actors may submit information on the desired characteristics of the model, i.e., model requirements. Thus, by personally taking part in the modelling process, they are able to influence the goals of the modelling and the means used. This power enables the actors to steer the modelling for their own benefit. Hence, they are more likely to commit to the goals and help the modellers in any way they can, i.e., they are motivated to constructing the model. This is a two-way relationship. On the one hand, the motivation of the process actors is crucial for participation. On the other, participation itself can be used to motivate the actors to influence the modelling outcome. Such actor influence on the outcome of the modelling is deemed important also by Humphrey (1995, p. 16). He argues that process actors must participate in process definition to help ensure the processes fit their personal needs as closely as possible.

It is of course conceivable that one or some of the participating actors dominates the others and misuses his or their power by directing the modelling for personal benefit. This emphasises the role of the modellers in controlling the modelling process and maintaining democracy in decision making. We experienced no need for such restricting, but this may be a serious risk in a less democratic and solidary work environments.

It is conceivable, that the advantages of participation could be utilised more widely in a modelling process than only in the model construction phase and in a model review. In fact, our interviews can be seen as a participative form of carrying out the interviews, since the interviewees had a role in leading the conversation. Another data collection method, participant observation, also enabled the actors to affect on our doings. By contrast, PML selection and adaptation was conducted without any direct actor involvement, which may be deemed as a neglect. Presumably, the PML we selected and moulded affected the actors' perception of their process model and thereby the model requirements. The pilot process model may have directed them to think through the PML, use its concepts and define their wishes in its terms. Hence, the PML may readily have limited the actors' process thinking. Such influence could perhaps be amended by creating several alternative models with different PMLs, based on which the actors could select a PML they prefer. This kind of procedure would, of course, require a higher level of process thinking from the actors and slightly more modelling effort than a PML selection carried out solely by the modellers, as was done in our case.

Having arrived to the end of the modelling process, we now recall the context in which this process was executed and reflect upon its effect on our experience.

7.5.4 Effect of Maidenly Context

We assume our modelling experience to be applicable to any maidenly process modelling context where understanding and communication of the software process is to be facilitated. The effect of the latter of these two presuppositions is evident. This is the modelling objective that our software process modelling principles are intended to serve and its influence on the modelling is discussed extensively above.

The first condition, by contrast, deserves some more attention. In our case, maidenly context means that there only existed a life-cycle-level description of the software process we modelled. This is, in fact, the reason for the need to facilitate understanding and communication. Besides the modelling aspect, our process context possessed other, more general, maidenly features as well. Namely, the process had not yet reached its final phases, several of the actors had only recently entered the process and the tools used represented state of the art technology. These two circumstances had a major impact on our study.

First, as the software process had not been modelled below the life-cycle level of detail, no PML had been selected or developed either. Thus, we could freely utilise the principle of PML engineering with no restrictions of any previous work.

Second, the virginity was experienced through the poor level of process documentation and continuously increasing amount of process data. As the current project progressed, more process information became available. Thus, the role of process data collection was emphasised. In our case, the data collection continued throughout the entire modelling case. After the most intensive data collection, during initial modelling, we interrogated the process actors and in participative modelling the actors themselves contributed the fresh details to the model.

It seems likely that the produced software process model would have been more detailed and formal if the software process had been executed repeatedly. Accordingly, the created model will probably have to be changed as the process matures.

In addition to these findings related to the principles and their application, the modelling life-cycle, which provided the framework for their application, offers some themes for discussion. These are focused on in the next section.

7.6 About Modelling Life-Cycles – Discussion

In Chapter 5 the principles were realised in a sequence of activities, which we simply denoted as a software process modelling process. It allowed for an abstract design for the modelling. The modelling case in the previous chapter implemented this process. In the course of the implementation, some thoughts arose regarding the form of the life-cycle and the level of detail of such a life-cycle. These are subjects contemplated in the following two subsections.

7.6.1 Alternative Modelling Life-Cycle

In our study, the principles were applied in a waterfall-life-cycle with four steps. Another conceivable approach might have been to use a spiral-life-cycle. Arguments for using the spiral model as a basis arise from our experience of the importance of prototyping (piloting) and the late attainment of the model requirements, which caused the interviews to result in a notable amount of redundant data.

The first round of the spiral-model would be dedicated to selecting and adapting the techniques as well as collecting the model requirements. On the second round, again, actual process data would be collected and the model constructed. Based on the outcome of this second round, any number of further iterations could be executed to develop the model. This presents a clear advantage over the waterfall-life-cycle. Moreover, the overlapping of the steps might be extended and made more explicit in the spiral-life-cycle. Still, the actual order of the steps is perceived as the same as in the waterfall-life-cycle we utilised. The single possibility for a deviation is that initial modelling might be conducted before the intensive data collection phase if there is enough process information available.

In retrospect, it appears that a spiral-life-cycle would have served our purpose better than the waterfall one did. Therefore, we recommend that any similar modelling situation be tackled through applying the principles in a spiral-life-cycle.

7.6.2 Level of Detail of a Modelling Life-Cycle

In the course of our modelling case, some thoughts arose regarding the desirable properties of a modelling life-cycle and more generally concerning the control of a modelling process. The term modelling life-cycle is here used to denote the course of a modelling process ranging from data collection to the actual modelling and possibly also further to the maintenance of the model. Such a life-cycle may be provided by a modelling method or, as in our case, it may be a more informal construct listing modelling activities.

During our modelling, we had to continuously develop our own techniques based on a vague idea of the wanted result and experiences presented in literature. The existing software process modelling methods provided no relief for this work. A detailed life-cycle would have been in place to guide us, inexperienced process modellers, in our effort. Thus, it can be concluded that in the case of unpractised modellers, a more specific life-cycle appears to be in place especially if the modelling objectives are clear. Then the modellers are able to select a life-cycle that best seems to suit their goals.

Should the modellers already possess process modelling experience, a higher-level modelling life-cycle may well suffice. Based on their experience, the modellers are capable of applying the abstract guidelines of the life-cycle in practice. This allows a significant amount of flexibility when compared to a more detailed life-cycle. In fact, lack of flexibility might be a major deficiency of a low-level modelling life-cycle. Specific rules, techniques and procedures leave little room for adaptation. Thus, a rudimentary higher-level modelling life-cycle could be the correct choice in a modelling case where the objectives are vague.

Conceivably, the combination of inexperienced modellers and ambiguous modelling objectives – as in our case – would best be served by high-level guidelines associated with a library of low-level rules, techniques and procedures, which would be based on previous modelling experience. Following these guidelines, the modellers could select those techniques that best suit their needs. The advantages of such flexibility are acknowledged by Harmsen (1997). He speaks of method fragments, building blocks which are used to build a situational software engineering method.

Our fundamental software process modelling principles attempt to provide high-level guidelines for such situational method building. They are to be applied in any modelling case where communication and understanding of a software process is to be facilitated in a maidenly context. The experience report offered, again, is to guide in the application of these guidelines and to offer techniques for the modelling. In Harmsen's terms, our experience is intended to provide some method fragments for software process modelling.

8 Conclusions and Future Work

This study has focused on software process modelling and providing methodical, non-technological, support for it. Its goal was to find issues beneficial for modelling a software process to facilitate understanding and communication in a context, where the process has not previously been modelled.

Investigating four problems. The goal has been striven by investigating four problems. The first problem was to a acquire a general understanding of software engineering and software process modelling. Then, we were to scrutinise software process modelling methods. After that, the subject of desirable issues to consider while modelling a software process was pondered. Finally, the last problem dealt with the use of these desirable issues in software process modelling practice.

The first question was concentrated on in Chapters 2 and 3. As a result, software engineering is concluded to denote an industrial domain and a field of science developing software and attempting to bring discipline to that work.

Although many efforts, including software engineering methods, CASE tools and life-cycle models, have promoted the domain a great deal during the past three decades, software engineering is still in its infancy. Recently, software process modelling, a field of software process engineering, has began to deal with this issue. It is a domain perceiving the software engineering process and depicting it in process models. Such modelling is involved in the methods mentioned, and it is also used by various assessment frameworks and improvement approaches. Traditionally, most modelling has been used to automate the software process, but now there is increasing recognition of understanding as the first goal. To date, much of the research emphasis in software process modelling has been on the development of modelling languages, most of which are formal.

The second problem stated, the state of the method support for software process modelling, was treated in Chapter 4. Generally, this area has received scantly attention, but it likely to change in the future. We found that methods are implicitly included in many process-centred software engineering environments (PSEEs), which, however, deal with the process capture itself only superficially and their methods and modelling languages are regrettably inflexible. Then, we discovered some PSEE-independent modelling methods providing a general life-cycle description of a modelling process. They also appreciate the understanding of the process context and select the modelling language in a flexible manner. Unfortunately, none of the methods in our selection address the issue of adapting the language(s) used or the modelling method itself. For the most part, they also neglect the use of process actor involvement in modelling.

The third issue raised was focused on in Chapters 5 and 7. Addressing the shortcomings of the modelling methods scrutinised and utilising their assets, Chapter 5 listed four issues to be taken into account when modelling a software process for the first time to facilitate understanding and communication. In Chapter 7 these principles were specified based on the modelling case.

The fundamental modelling principles are flexibility, multiple data sources, PML engineering and participation. They suggest that the modelling should be bound to the process context, it should be based on data from multiple sources, the PML should be modified accordingly and the process actors should be involved in the modelling.

The fourth problem was to apply the principles in practice. This was also tackled in Chapters 5, 6 and 7. First, in Chapter 5, we demonstrated their application by presenting one feasible modelling life-cycle and then, in Chapter 6, this modelling process was executed in an industrial modelling case. Finally, in Chapter 7, some conclusions regarding the implementation were made based on our experience.

First, we found process documentation study, participant observation, interviews, and queries to be adequate data collection techniques and conceptual modelling useful in moulding the PML. Second, our experience shows that flexibility should be applied in all of the modelling process. Third, a spiral-like modelling life-cycle is conceived to be appropriate in applying the principles.

Implications. We hypothesise that the principles be applicable to any modelling situation similar to the one which was under scrutiny in this study. Our results suggest that applied according to the process context, the principles provide considerable assistance in facilitating understanding and communication in a maidenly context. Thus, we particularly emphasise the importance of PML engineering during the modelling process and actor participation in the modelling. These issues have been largely ignored in current research.

Future work. Our findings clearly indicate the need to study more the methodological aspects of software process modelling, especially modelling methods and modelling method and language engineering. Theory-oriented research alone would, however, alienate us from the practical life of software engineering. Thus, further research is required in regard to using the modelling methods in realistic modelling situations.

Pertaining more specifically to this study, it is necessary to validate the principles by applying them to several modelling situations. Our one modelling case shows only some major directions, but offers no real generalisable results. Experience gained in using the principles might be collected and fused into a proper modelling method. Especially, the principles of PML engineering and actor participation as well as participant observation as a data collection technique have to be more thoroughly contemplated and experimented in various settings. We had no resources for an in depth study focusing on each sub-area of process modelling that was encountered. Moreover, the set of principles should be extended to cover process model maintenance and further utilisation in software process improvement, which we left out of our scope.

In addition to these, we perceive a huge demand for software process modelling experience reports in general. These are to give practical feedback of how software process modelling can be supported by methods, method and language engineering and participation.

Experiences of realistic industrial modelling situations also transmits the true needs of those contexts to the modelling research domain. We believe that this would re-direct the research. The current modelling research has excessively overemphasised the automation objectives of software process modelling seeking a shortcut to happiness. Much of these results are only usable in sophisticated modelling contexts where process improvement has already progressed a great deal. Instead, the very beginning of the improvement life-cycle, understanding of the process, should be focused on. We expect these results to be far more widely applicable in the industrial software engineering field.

REFERENCES

Aoyama, M. 1993. Concurrent development process model. *IEEE Software*. 10(4): 46-55.

Armenise, P., Bandinelli, S., Ghezzi, C., Morzenti, A. 1993. A survey and assessment of software process representation formalisms. *International Journal of Software Engineering and Knowledge Engineering* 3(3): 401-426.

Armitage, J. W., Briand, L., Kellner, M. I., Over, J. W., Phillips, R. W. 1994. *Software process definition guide: Content of enactable software process representations, CMU/SEI-94-SR-21*. Pittsburgh: Software Engineering Institute.

Armitage, J. W., Kellner, M. I. 1994. A conceptual schema for process definitions and models. In *Proceedings of the 3rd international conference on the software process*. Washington: IEEE Computer Society Press. 153-165.

Auramäki, E., Leppänen, M., Savolainen, V. 1988. Universal framework for information activities. *Data Base*, ACM 19(1): 11-20.

Avison, D., Wood-Harper, A. 1990. *Multiview: An exploration in information systems development*. Alfred Waller Ltd. 129-143.

Avrilionis, D., Belkhatir, N., Cunin, P. 1996. A unified framework for software process enactment and improvement. In *Proceedings of the 4th international conference on the software process*. Brighton: IEEE Computer Society Press. 102-111.

Bandinelli, S. C., Fuggetta, A., Ghezzi, C. 1993. Software process model evolution in the SPADE environment. *IEEE Transactions on Software Engineering* 19(12): 1128-1144.

Bandinelli, S., Fuggetta, A., Lavazza, L., Loi, M., Picco, G. P. 1995. Modeling and improving an industrial software process. *IEEE Transactions on Software Engineering* 21(5): 440-454. http://www.elet.polimi.it/section/compeng/se/swproc/public.html (3.9.1996).

Basili, V. R., Caldiera, G., Rombach, H.D. 1994. The experience factory. In *Anonymous encyclopedia of software engineering*. New York: John Wiley & Sons. 470-476. http://www.iese.fhg.de/Publications/ESEG.html (25.3.1997).

Basili, V. R., Rombach, H. D. 1988. The TAME project: Towards improvement-oriented software environments. *IEEE Transactions on Software Engineering* 14(6): 758-772.

Basili, V. R., Rombach, H. D. 1987. Tailoring the software process to project goals and environments. In *Proceedings of the 9th international conference on software engineering*. IEEE Computer Society Press. 345-357.

Benington, H. D. 1956. Production of large computer programs. In *Proceedings of ONR Symp. Advanced Programming Methods for Digital Computer*. (June):15-27.

Blum, B. I. 1994. A Taxonomy of software development methods. *Communications of the ACM* 37(11): 82-94.

Boehm, B.W. 1981. *Software engineering economics*. Prentice-Hall.

Brinkkemper, S. 1990. *Formalisation of information systems modelling*. Amsterdam: P.h. thesis publishers.

Brinkkemper, S., Harmsen, F., Oei, H. 1995. Configuration of situational process models: An information systems engineering perspective. In *Proceedings of the european workshop on software process technology (EWSPT'95)*, ed. Schaefer, W. Lecture Notes in Computer Science, LNCS 913, Springer-Verlag. 193-196.

Brooks, F. 1987. No silver bullet: Essence and accidents of software engineering. *IEEE Computer* 20(4): 10-19.

Budde, R., Kautz, K., Kuhlenkamp, K., Züllighoven, H. 1992. *Prototyping – An approach to evolutionary systems development.* Berlin: Springer-Verlag.

de Bunje, T., Engels, G., Groenewegen, L., Matsinger, A., Rijnbeek, M. 1996. Industrial maintenance modelled in SOCCA: An experience report. In *Proceedings of the 4th international conference on the software process.* IEEE Computer Society Press. 13-26.

Burns, R. N., Dennis, A. R. 1985. Selecting the appropriate development methodology. *Data Base* (Fall): 19-23.

Christie, A., M. 1993. A graphical process defintion language and its application to a maintenance project. *Information and Software Technology* 25(6/7): 364-374.

Coad, P., Yourdon, E. 1991a. *Object-oriented design* (ISBN 0-13-629981-4). New Jersey: Prentice-Hall.

Coad, P., Yourdon, E. 1991b. *Object-oriented design* (ISBN 0-13-630070-7). Prentice-Hall.

Conradi, R., Fernström, C., Fuggetta, A., Snowdon, R. 1992. Towards a reference framework for process concepts. In *Proceedings of the 2nd european workshop on software process technology.* Springer-Verlag. 2-17.

Conradi, R., Fernström, C., Fuggetta, A. 1993. A conceptual framework for evolving software processes. *ACM SIGSOFT Software Engineering Notes* 18(4): 26-35.

Conradi, R., Hagaseth, M., Larsen, J-O., Nguyen, M. N., Munch, B. P., Westby, P. H., Zhu, Weicheng, Jaccheri, M. L., Liu, C. 1994a. EPOS: Object-oriented and cooperative process modelling. In *Software process modelling and technology,* ed. A. Finkelstein, J. Kramer and B. A. Nuseibeh (ISBN 0-86380-169-2). 33-70.

Conradi, R., Høydalsvik, G. M., Sindre G. 1994b. A comparison of modelling frameworks for software processes and information systems. In *Proceedings of the 3rd european workshop on software process technology.* Springer-Verlag. 254-260. http://www.idt.unit.no/~epos/bibliografia.html (5.2.1997).

Constantine, L. L., Lockwood, L. A. D. 1993. Orchestrating project organisation and management. *Communications of the ACM* 36(10): 31-43.

Crosby, P. 1979. *Quality is free.* McGraw-Hill.

Curtis, B. 1992. The CASE for process. In *The impact of computer supported technologies on information systems development,* ed. K. E. Kendall et al. North-Holland: Elsevier Science Publishers B.V. 333-343.

Curtis, B. Kellner, M. I., Over, J. 1992. Process modelling. *Communications of the ACM* 35(9): 75-90.

Curtis, B., Krasner, H., Iscoe, N. 1988. A field study of the software design process for large systems. *Communications of the ACM* 31(11): 1268-1287.

Curtis, B., Krasner, B., Shen, V., Iscoe, N. 1987. On building software process models under the lamppost. In. *Proceedings of the ninth international conference on software engineering.* Washington, DC: IEEE Computer Society. 96-103.

Curtis, B., Paulk, M. 1993. Creating a software process improvement program. *Information and software technology* 35(6/7): 381-386.

Dandekar, A., Perry, D. E., Votta, L. G. 1996. A study in process simplification. In *Proceedings of the 4th international conference on the software process.* Brighton: IEEE Computer Society Press. 27-35.

Deiters, W., Gruhn, V. 1994. The funsoft net approach to software process management. *International Journal of Software Engineering and Knowledge Engineering* 4(2): 229-256.

Dorling, A. 1993. SPICE: Software process improvement and capability determination. *Information and Software Technology* 35(6/7): 404-406.

Dowson, M. 1987. Integrated project support with IStar. *IEEE Software* (November): 6-15.

Dutton, J. E. 1993. Commonsense approach to process modeling. *IEEE Software* (July): 56 - 64.

Ellmer, E. 1995. Improving software processes. In *Proceedings of the 7th conference on software engineering environments,* ed M.S. Verrall. IEEE Computer Society Press. 74-83.

Ehn, P., Kyng, M. 1987. The collective resource approach to systems design. In *computers and democracy: A scandinavian challenge.* ed. G. Bjerknes, P. Ehn and M. King. Brookfield, Vermont: Avebury Gower.

Emmerich, W., Bandinelli, S., Lavazza, L., Arlow, J. 1996. Fine grained process modelling: An experiment at British Airways. In *Proceedings of the 4th international conference on the software process.* Brighton: IEEE Computer Society Press. 2-12.

Engels, G., Groenewegen, L. 1994. SOCCA: Specifications of coordinated and cooperative activities. In *Software process modelling and technology,* ed. A. Finkelstein, J. Kramer and B. A. Nuseibeh (ISBN 0-86380-169-2). 71-102.

Finkelstein, A., Kramer, J., Nuseibeh, B., eds. 1994. *Software process modelling and technology.* Advanced Software Development Series, Research Studies Press Ltd. (John Wiley) (ISBN 0-86380-169-2).

Forte, G., Norman, R. J., 1992. A self-assessment by the software engineering community. *Communications of the ACM* 35(4): 28-32.

Haikala, I., Märijärvi, J. 1995. *Ohjelmistotuotanto* (Software engineering in Finnish). Jyväskylä: Suomen ATK-kustannus Oy.

Harel, D. 1992. Biting the silver bullet: Toward a brighter future for system development. *IEEE Computer* 25(1): 8-20.

Harmsen, A. F. 1997. *Situational method engineering.* Doctoral dissertation, University of Twente. Utrecht, Netherlands: Moret Ernst & Young Management Consultants.

Heineman, G. T., Botsford, J. E., Caldiera, G., Kaiser, G. E., Kellner, M. I., Madhavji, N. H. 1994. Emerging technologies that support a software process life cycle. *IBM Systems Journal* 33(3): 501 - 529.

Herbsleb, J. D., Goldenson, D. R. 1996. A systematic survey of CMM experience and results. In *Proceedings of the 18th international conference on software engineering.* IEEE Computer Society Press.

Heym, M., Österle, H. 1993. Computer-aided methodology engineering. *Information and Software Technology* 35(6/7): 345-354.

Hirschheim, R., Klein, H. 1992. Paradigmatic influences on information systems development methodologies: Evolution and conceptual advances. *Advances in Computers* (ISBN 0-12-012134-4), 34: 293-392.

Hughes, J. K, King, V., Rodden, T., Anderson, H. 1994. Moving out from the control room: Ethnography in system design. In *Proceedings of the conference on computer-supported cooperative work.* New York: Association for Computing Machinery. 429-439.

Humphrey, W. 1988. Characterizing the software process: A maturity framework. *IEEE Software* (March): 73-79.

Humphrey, W. 1989. *Managing the software process.* Addison-Wesley Publishing Company.

Humphrey, W. 1995. *A discipline for software engineering.* Addison-Wesley Publishing Company.

Höltje, D., Madhavji, N. H., Bruckhaus, T., Hong, W. K. 1994. Eliciting formal models of software engineering processes. In *Proceedings of the 1994 CAS conference (CASCON'94).* IBM Canada Ltd. and The National Research Council of Canada. http://softeng.cs.mcgill.ca/~till/publications.html (7.3.1997).

Iivari, J. 1996. Why are CASE tools not used. *Communications of the ACM* 39(10): 95-103.

Jarke, M., Pohl, K. 1992. Information systems quality and quality information systems. *The impact of computer supported technologies on information systems development,* ed. K.E. Kendall et al. North-Holland: Elsevier Science Publishers B.V. 345-375.

Jorgensen, P. C. 1990. Accelerating process maturity with CASE. *American Programmer.* (Septemper): 10-15.

Junkermann, G. Peuschel, B., Schäfer, W., Wolf, S. 1994. MERLIN: Supporting cooperation in software development through a knowledge-based environment. In *Software process modelling and technology,* ed. A. Finkelstein, J. Kramer and B. A. Nuseibeh (ISBN 0-86380-169-2). 103-131.

Kaiser, G. E., Ben-Shaul, I. Z., Popovich, S. S. 1996. A metalinguistic approach to process enactment extensibility. In *Proceedings of the 4th international conference on the software process.* Brighton: IEEE Computer Society Press. 90-101.

Kaiser, G. E., Popovich, S. S., Ben-Shaul, I. Z. 1993. A bi-level language for software process modeling. In *Proceedings of the 15th international conference on software engineering.* IEEE Computer Society Press. (May): 132-143.

Kehoe, R, Jarvis, A. 1995. *ISO-9000-3: A tool for software product and process improvement.* New York: Springer-Verlag.

Kellner, M. I., Briand, L., Over, J. W. 1996. A method for designing, defining, and evolving software processes. In *Proceedings of the 4th international conference on the software process.* Brighton: IEEE Computer Society Press. 37-48.

Kellner, M. I., Hansen, G. A., 1988. Software process modeling. *Technical Report CMU/SEI-88-TR-9.* Software Engineering Institute, Carnegie Mellon University. ftp://ftp.sei.cmu.edu/pub/documents/88.reports/pdf/tr09.88.pdf (10.2.1997).

Kellner, M. I., Hansen, G. A., 1989. Software process modeling: A case study. In *Proceedings of the 22nd annual Hawaii international conference on systems sciences.* IEEE Computer Society Press. 2(January): 175-188.

Kemerer, C. F. 1992. How the learning curve affects CASE tool adaption. *IEEE Software* 9(3): 23-28.

King, S., Galliers, R. 1994. Modelling the CASE process: Empirical issues and future directions. *Information and Software Technology* 36(10): 587-596.

Kitson, D. H., Masters, S. M. 1993. An analysis of SEI software process assessment results: 1987-1991. In *Proceedings of the 15th international conference on software engineering.* IEEE Computer Society Press. 68-77.

Klingler, C. D., Schwarting, D. 1994. A practical approach to process definition. In *Proceedings of the 7th annual software technology conference.* http://source.asset.com/stars/darpa/Papers/ProcessDDPapers.html (18.2.1996).

Koch, G. R. 1993. Process assessment: The 'BOOTSTRAP' approach. *Information and Software Technology* 35(6/7): 387-403.

Kontio, J. 1994. *Promises: A framework for utilizing process models in process asset management.* Licentiate thesis at Helsinki University of Technology.

Koskinen, M. 1996. *Bringing process concepts alive: On designing process modelling languages in a metacase environment.* Master's thesis at the University of Jyväskylä.

Koskinen, M., Marttiin, P. 1997. Process support in MetaCASE: implementing the conceptual basis for enactable process models in metaEdit+. In *Proceedings of the 8th Conference on Software Engineering Environments.* IEEE Computer Society Press. 110-122,

Krasner, H., Terrel, J., Linehan, A., Arnold, P., Ett, W. H. 1992. Lessons learned from a software process modeling system. *Communications of the ACM* 35(9): 91-100.

Kumar, K., Welke, R. J. 1992. Methodology engineering: A proposal for situation-specific methodology construction. In *Challenges and strategies for research in systems development,* ed. W. W. Cotterman and J. A. Senn. John Wiley and Sons Ltd. 257-269.

Lai, R. 1993. Mature process. *IEEE Software* (July): 14-17.

Lonchamp, J. 1993. A structured conceptual and terminlogical framwork for software process engineering. In. *Proceedings of the 2nd international conference on the software process.* IEEE Computer Society Press. 41-53. http://www.loria.fr/~jloncham/papers.html (5.2.1997).

Lonchamp, J. 1994a. A collaborative process-centered environment kernel. *In Proceedings of the 6th international conference on computer aided infomation systems engineering*, ed. G. Wijers, A. I. Wasserman. Lecture Notes in Computer Science Springer-Verlag. 811: 28-41.

Lonchamp, J. 1994b. An assessment exercise. In *Software process modelling and technology,* ed. A. Finkelstein, J. Kramer and B. A. Nuseibeh (ISBN 0-86380-169-2). 335-356.

Lott, C. M. 1993. Process and measurement support in SEEs. In *ACM SIGSOFT software engineering notes.* 18(4): 83-93.

Lott, C. M., Rombach, H. D. 1993. Measurement-based guidance of software projects using explicit project plans. *Information and Software Technology* 35(6/7): 407-419.

Madhavji, N. H. 1991. The process cycle. *Software Engineering Journal* (September): 234-242.

Madhavji, N. H., Höltje, D. Hong, W. K., Bruckhaus, T. 1994. Elicit: A method for eliciting process models. In *Proceedings of the 3rd international conference on software process.* IEEE Computer Society Press. http://softeng.cs.mcgill.ca/~till/publications.html (7.3.1997)

Marttiin, P. 1994. Towards flexible process support with a CASE shell. In *Advanced information systems engineering,* ed. G. Wijers, S. Brinkkemper, and T. Wasserman. Springer-Verlag. 14-27.

Marttiin, P., Harmsen, F., Rossi, M. 1996. A functional framework for evaluating CAME environments: The case of Maestro II / Decamerone and MetaEdit+ assessment. *IFIP WG 8.1/8.2 Working Conference on Principles of Method Construction and Tool Support,* Atlanta, USA. (August): 63-86.

Marttiin, P., Lyytinen, K., Rossi, M., Tahvanainen, V.-P., Smolander, K., Tolvanen, J.-P. 1995. Modeling requirements for future CASE: Issues and implementation considerations. *Information Resources Management Journal* 8(1): 15-25.

Marttiin, P., Rossi, M., Tahvanainen, V-P., Lyytinen, K. 1993. A comparative review of CASE shells: A preliminary framework and research outcomes. *Information & Management* 25: 11-31.

McChesney, I. R. 1995. Towards a classification scheme for software process modelling approaches. *Information and Software Technology* 37(7): 363-374.

McFeeley, B. 1996. IDEAL: *A guide for software process improvement. CMU/SEI-94-SR-21.* Pittsburgh: Software Engineering Institute.

Nguyen, M. N., Conradi, R. 1994. The Software meta-process: Taxonomy and assessment. In *Proceedings of the 3rd international conference on software process (ICSP'3).* IEEE Computer Society Press. 10-11: 167-175. http://www.idt.unit.no/~epos/bibliografia.html (12.7.1996).

Nunamaker, J. F. jr., Chen, M., Purding, T. D. M. 1991. Systems development in information systems research. *Journal of Management Information Systems* 7(3): 89-106.

Nutt, G. J. 1995. Software engineering process model – a case study. In *Proceedings of the 1995 ACM conference on organizational computing systems (COOCS'95)*, ed. N. Comstock and C. A. Ellis. 130-137.

Olerup, A. 1989. Socio-technical design of computer-assisted work: A discussion of the ETHICHS and Tavistock approaches. *Scandinavian Journal of Information Systems*. 1: 43-71.

Orlikowski, W. J. 1993. CASE tools as organisational change: Investigating incremental and radical changes in systems development. *MIS Quarterly* (September): 309-340.

Osterweil, L. 1987. Software processes are software too. In *Proceedings of the 9th international conference on software engineering*. IEEE Computer Society Press. 2-13.

Paulk, M. C., Curtis, B., Chrissis, M. B. 1993. Capability maturity model, version 1.1. *IEEE Software* (July): 18-27.

Paulk, M. C. Weber, C. V. Curtis, B. Chrissis, M. B. 1995. *The capability maturity mModel: guidelines for improving the software process*. Addison-Wesley Publishing Company.

Pérez, G, El Emam, K., Madhavji, N. H. 1996. Evaluating the congruence of a software process model in a given environment. In *Proceedings of the 4th international conference on the software process*. Brighton: IEEE Computer Society Press. 49-62.

Pohl, K. 1996. *Process-centered requirements engineering*. Advanced Software Development Series, Research Studies Press Ltd., John Wiley & Sons, Inc.

Pressman, Roger S. 1992. *Software engineering: A practitioner's approach*. 3rd ed. Berkshire, England: McGraw-Hill, Inc.

Proceedings of the 18th international conference on software engineering. 1996. Berlin, Germany: IEEE Computer Society Press.

Proceedings of the 9th international conference on software engineering education. 1996. Daytona, USA: IEEE Computer Society Press.

Proceedings of the 4th international conference on the software process. 1996. Brighton: IEEE Computer Society Press.

Proceedings of the 5th european workshop on software process technoloqy. 1996. Lecture Notes in Computer Science, Nancy, France: Springer-Verlag.

Proceedings of the 10th international software process workshop (ISPW10). 1996. Ventron. France.

Proceedings of the 1st european workshop on software process modelling. 1991. Milan, Italy.

Proceedings of the 6th international conference on software quality (6ICSQ). 1996. Ottawa, Canada.

Proceedings of the 5th european conference on software quality (5ECSQ). 1996. Dublin, Ireland: European Organization for Quality - Software Committee.

ProSLCSE Project. *Visual process modelling language*. http://www.issi.com/proslcse-3.5i/vmpl.html (25.3.1997).

Raccoon, L., 1995a. The chaos model and the chaos life cycle. *ACM SIGSOFT Software Engineering Notes* 20(1): 55-67.

Raccoon, L., 1995b. The complexity gap. *ACM SIGSOFT Software Engineering Notes* 20(3): 37-43.

Rossi, M. 1995. The MetaEdit CAME environment. In *Proceedings of MetaCASE -95.* Sunderland University Press. January 5-7.

Royce, W. 1970. Managing the development of large software systems. In *Proceedings of IEEE Wescon.* New York: IEEE Computer Society Press. 1-9. (Reprinted in *Proceedings of the 9th ICSE.* 1987. IEEE Computer Society Press. 328-338.)

Rumbaugh, J. et al. 1991. *Object-oriented modelling and design.* Prentice Hall.

Schwartzman, H. B. 1993. *Ethnography in organisations.* Qualitative Research Methods Series 27, Newbury Park, United State of America: Sage Puclications, Inc.

Shepard, T., Wortley, C., Sibbald, S. 1992. A visual software process language. *Communications of the ACM* 35(4): 37-44.

Smolander, K. 1991. OPRR, A Model for modelling systems development methods. *Next generation CASE tools*, ed. K. Lyytinen and V.-P. Tahvanainen. Amsterdam, the Netherlands: IOS Press.

Starke, G. 1993. Urgent research issues in software process engineering. *ACM SIGSOFT Software Engineering Notes* 18(4): 13-15.

Swenson, K. D., Maxwell, R. J., Matsumoto, T., Saghari, B., Irwin, K. 1994. A business process environment supporting collaborative planning. *The Journal of Collaborative Computing* 1(1): 15-34.

Tamai, T. 1993. Current practices in software processes for system planning and requirements analysis. *Information and Software Technology* 35(6/7): 339-344.

Totland, T., Conradi, R. 1995. A survey and comparison of some research areas relevant to software process modeling. In *Proceedings of the 5th european workshop on software Process technology.* http://www.pakt.unit.no/~tt/research/pub.htm (18.2.1997).

Turgeon, J., Madhavji, N. H. 1996. A systematic, view-based approach to eliciting process models. In *Proceedings of the 5th european workshop on software process technology.* Springer-Verlag.

Vessey, I., Sravanapudi, A. P. 1995. CASE tools as collaborative support technologies. *Communications of the ACM* 38(1): 83-95.

Yin, R. K. 1994. *Case study research: Design and methods.* 2nd ed. Applied Social Research Methods Series, Sage Publications.

Yourdon, E. 1989. *Modern structured analysis.* Yourdon Press.

Yu, E. S. K., Mylopoulos, J. 1994. Understanding "why" in software process modelling, analysis, and design. In *Proceedings of the 16th international conference on software engineering.* Sorrento, Italy: IEEE. 159-168.

Appendix A: EXAMPLE NOTICE

The purpose of this notice was to tell the process actors about the modelling process. It was distributed among them during the beginning stage of our research.

Attention:

The purpose of this notice is to tell what we plan to do during the next six months. As in the last group meeting was told we are doing our final thesis here among you. The topic of our thesis is software process modelling. Through that we mean to establish a better understanding of the software process.

Next follows: first the goals of the research, second the time schedule and last what this all has to do with you.

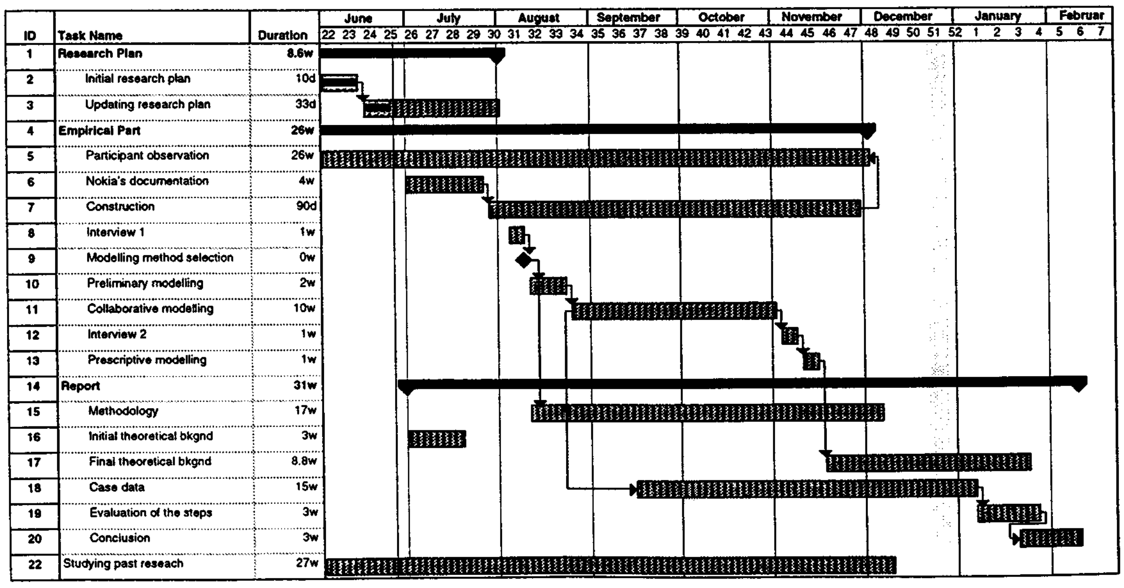
Goals (freely from our Research Plan):

The research problem is: “How to use software process modelling to make a software process visible among its actors?" The actors - that's you.

So our goal is to capture characteristics of the current software process, with your help, and improve your understanding of it by using process modelling and especially collaborative modelling. By capturing your experiences of the current process we hope to help the execution of future processes.

We expect the study to result in a model of the current software process, in a shared understanding of the process among you and general guidelines for the continuation of our work.

Tasks and schedule:



ID	Task Name	Duration
1	**Research Plan**	**8.6w**
2	Initial research plan	10d
3	Updating research plan	33d
4	**Empirical Part**	**26w**
5	Participant observation	26w
6	Nokia's documentation	4w
7	Construction	90d
8	Interview 1	1w
9	Modelling method selection	0w
10	Preliminary modelling	2w
11	Collaborative modelling	10w
12	Interview 2	1w
13	Prescriptive modelling	1w
14	**Report**	**31w**
15	Methodology	17w
16	Initial theoretical bkgnd	3w
17	Final theoretical bkgnd	8.8w
18	Case data	15w
19	Evaluation of the steps	3w
20	Conclusion	3w
22	Studying past reseach	27w

Figure 1. Time schedule of the research.

What this all has to do with you

If you are really in hurry read only the paragraphs beginning with bold text.

Participant observation means we shall be here with you during our research hanging around and making questions. We also shall build a module of software, which is a part of Direct Mode. The times given below are preliminary and will be specified after the vacations.

Interview 1 (about one hour per person during one week). The purpose of the first interview is to supply us information concerning the current software process of SW(D)-team. The interview will consist of questions such as "Who does what?", "By which means?", and "Why?".

We shall select or build a method for modelling the software process according to the results of the first interview. We shall also create a base model of the software process based on the information collected from the first interview and other sources.

Collaborative modelling (about five to eight hours per person during ten weeks). We intend to continue modelling in cooperation with you. The task aims at a shared view of the process by describing the process' present state in as great detail as needed considering the dynamic nature of the process. Modelling will concentrate on the most essential aspects discovered from the first interview. Suggestions for what could be done better are expected and they will be modelled separately from the main model.

Collaborative modelling is the most essential part of our research and we hope you will help us with it by taking part and by cooperating with us, after all the purpose of this modelling is to help you in your future projects.

Interview 2 (about two to five hours per person during one week) The second interview will consist of two parts - regular interview and a common session - first of which is optional. The purpose of the interview is to find out how well the modelling has succeed how well you are aware of the process after modelling. The purpose of the common session is to visualise future directions in the light of findings from the modelling and interviews.

After the interview we intend to build a prescriptive model to prescribe the process for the future according to the findings obtained in the previous phase.

"Press release" follows at the beginning of the august, in other words we have a meeting where we tell more about the our plans.

Regards from research guys.

Appendix B: Interview Theme List

This document represents the final development stage of the theme list used in our interviews. The purpose of the list was to guide the interviewer by providing him with the interview themes and some refining questions.

1 Background

- Do you know why we have this interview?
- How long have you been in SW(D) group?
- What kind of picture did you have of SW(D) when you started?
- Did you have any training or like when you started?

2 Themes

- What have been your tasks?
- Who gave you the tasks, with whom did you work and when?
- How did you do the task?

3 Coordination

- Who gave you the order to do the task and did anyone guide at the beginning?

4 Tools

- How about the learning of SDT, any training/guidance or by your self

5 Communication

- While doing this task did you communicate with other members of the SW(D), and if you did about what and why?
- With who?
- How?
- Formality?
- Sufficiency?

6 Information needs and their fulfilment

- How have you full filled your information needs i.e. how have you got the information you have needed?

7 Proceeding

- How did you proceed with your task, did you have any plan. Did you have to return to previous phases?

8 Problem solving

- Did you have any big trouble with the task and if you had how did you solve them, did you have any standard procedures?
- Did you need to cooperate?
- Sharing of solutions?

9 Rationale in tasks

- In you opinion why module description is created?
- What factors influenced into the proceeding, sequence etc.?

10 Critical factors

- What or which thinks were most important to complete the task?

11 Training

- Have you got any training for your tasks?
- Has it been adequate?

12 Documents required by ISO 9001 standard

- Are you familiar with: project plan, sw process description, version control documentation and dokumentointiohjeet. Have they had influence in your work?

13 Fluctuating requirement specifications

- Have you had problems with changing requirement specifications, what kind of problems and how solve them?

14 Application domain knowledge

- How well you know the application domain as a whole: Tetra, Nokia requirements and telecommunication.
- How did you get it and do you spread it?

15 The role of the meetings

- What have been the role of meetings for you?

16 Team

- What makes SW(D) working?

17 The role in the team

- In your opinion what is your role in SW(D)?

18 Future

- What will you do next, what is the future of your tasks, SW(D), etc.
- How the project will proceed?
- How the team will mature?

19 Process

- What is your current picture / impression of the sw process of SW(D)?

20 Modelling

- In your opinion, what should we do, what should we model and make more visible by that?

Appendix C: DRAFT INTERVIEW REPORT SAMPLE

The audio records taken in the interviews were first extracted to draft reports such as this one.

Interviewer: How did you start in D-group?

Interviewee: September 1995

Interviewer: When you started, what kind of a picture did you have of the overall software process of the group?

Interviewee: Not much of an idea. SDL in use, forward-looking in trying new methods, strong commitment to testing. Having a testing team is unusual.

Interviewer: So, you had previous experience with SDL

Interviewee: Used for modelling, not code generation. Not useful experience, more useful when used for code generation.

Interviewer: Did you have any training or guidance when you started?

Interviewee: Training courses (Basic SDL, code generation, Nokia OS), basic familiarising rumba.

Interviewer: Did they help you to get started?

Interviewee: Not much, came too early. This is true for a lot of people.

[...]

Interviewer: After the reading period you started to write the module description. How did it proceed? Who gave you that task?

Interviewee: Kimmo gave it to me (the tasks were given in discussion, in team meetings). He did not know exactly what I was supposed to do – Pasi wrote the first module description and it became a standard.

Interviewer: So, how did you proceed with this module description phase?

Interviewee: Matter of determining all the interfaces to communicating modules, partly by looking at TETRA-standard, partly by talking to the designers of the other modules that had interfaces with mine.

Interviewer: Did you have many interfaces to other modules?

Interviewee: 9. Some bigger that others (1-40 signals)

Interviewer: So, you had to communicate especially with those designers having interfaces to your module. How did that proceed?

Interviewee: No formal way. No formal meetings. Occasional questioning – Reading to parts of TETRA related to the module X, asking Pasi. – Meetings about the module X generally; how it should be implemented in Nokia/TETRA-system: Meetings with people in Tampere and Espoo.

Interviewer: What did you discuss with them?

Interviewee: The features of the module X. TETRA didn't give the exact implementation – Partly to design the user interface, partly to decide on implementation issues.

Interviewer: So, what was the role of these meetings to you?

Interviewee: To decide the functionality, partly to decide the interface to the SwMI.

Interviewer: So, while you were writing the module description, you used TETRA-standard and communicated with other module designers. Did you have any other information sources?

Interviewee: Few others, not great influence – Notes-database, but many things were not implemented. There was obviously some informal communication to some other departments. (Email to the base station people. "How are you going to implement this?")

Interviewer: So, you didn't have requirements, or what was the problem?

Interviewee: Basically the requirements are TETRA – The definition of some features was going on.

Interviewer: Did you have any big trouble writhing the module description?

Interviewee: At first not a clue what to do; additional features were not in TETRA. – No big problems, just the share volume of information.

Interviewer: Why, in your opinion, are the module descriptions written?

Interviewee: Good question – To have a good idea of what you are going to implement. The newness is unique in this project. Later beneficial for maintaining the system. Even if it is not entirely up to date, useful.

Interviewer: So you find them useful?

Interviewee: Currently not very. Later, when this project is going to end. – Maybe in testing.

Appendix D: E-MAIL QUERY

The document below presents the e-mail query sent to the process actors directly after the interviews. Its role was to validate the interview results by allowing everyone a chance to comment on all the model requirements arisen in the interviews.

Haastattelut on nyt pidetty ja purettu ja haastattelijoiden mielestä ne menivät oikein hyvin. Kiitokset teille aktiivisesta osallistumisesta ja kiinnostuksesta. Olemme keränneet lausunnoistanne vaatimuksia itse ohjelmistoprosessin mallinnukselle. Kuten toivottavasti suurin osa muistaa, osoittautui haastatteluissa hankalaksi nimenomaan kysymys koskien sitä, mitä meidän pitäisi mallintaa. Haluammekin nyt, että pohdiskelisitte vielä hetken kyseistä kysymystä, elikkäs:

Mihin meidän kannattaisi mallinnuksessa keskittyä, mitä prosessin aspekteja/tekijöitä tuoda esille ja miten? Mitä mallissa pitäisi olla?

Olemme tämän mailin loppuun koonneet yhteenvedon haastatteluissa saaduista vastauksista tähän kysymykseen. Lisäksi olemme hakeneet täydennystä muiden haastattelukysymysten vastauksista. Alaotsikoiden lopuissa oleva numero kertoo mielipiteen painon. Mielipidekokoelman on tarkoitus toimia virikkeenä eli lähtökohtana, jota voitte vapaasti kommentoida, jos olette eri eli samaa mieltä yhdestä tai useammasta kohdasta. Uudet ehdotukset ovat erityisen tervetulleita.

Elipäsiis: Jos suinkin viitsitte, niin:

1. kommentoikaa joko eri kohtia suoraan tai sitten kirjoittakaa mailin loppuun kommenttinne.

2. Rustailun päätteeksi lähettäkää maili kommentteineen replynä meille.

3. Antamanne tiedon perusteella osaamme sommitella yhteistyössä mallinnuksen pohjaksi prosessimallia, josta saadaan enemmän irti. Näin pyrimme ainakin minimoimaan meistä aiheutuvan haitan ja ehkä jopa tuottamaan jotakin hyödyllistä.

1 Yhteenveto mallinnuksen vaatimuksista

1.1 Yleiset vaatimukset mallille (eli "Millainen mallin tulisi olla?")

1.1.1 Yhteensopivuus (2)

Prosessimalli tulisi voida sovittaa tuoteprosessimalliin sekä sen tulisi täydentää/korvata nykyinen SW-prosessimalli.

1.1.2 Suunnittelun tuki (2)

Prosessimallin tulisi palvella projektin suunnittelua ja seurantaa, toimia ikään kuin tarkistuslistana siitä, mitä tarvitaan tehtävän suorittamiseksi. Mallista tulisi ilmetä tehtävien järjestys ja sen avulla pitäisi ajankäyttö pystyä suunnittelemaan.

1.1.3 Muita käyttötarkoituksia (1)

Prosessimallin pitäisi olla toimintamalli siitä, miten asioita tehdään se voisi myös toimia kommunikointivälineenä markkinointiin päin.

1.1.4 Geneerisyys (5)

Prosessimallin geneerisyys oli monien toivoma ominaisuus. Mallin ei tulisi olla liian yksityiskohtainen vaan sen kuuluisi olla selkeä ja yksinkertainen ottaen kuitenkin kehitysympäristön huomioon ja tuoden prosessin tärkeät osat näkyviin. Mallin pitäisi kuvata millainen prosessi on ollut ja millainen se tulee olemaan.

1.2 Mallinnettavat prosessin piirteet (eli "Mitä mallissa tulisi olla?")

1.2.1 Aktiviteetit, syötteet ja tulosteet (6)

Toiminnot ja vaihetuotteet ovat nykyisessä mallissa sekaisin, kehitettävässä nämä pitäisi erotella selkeämmin. Mallista tulisi näkyä vaiheiden/aktiviteettien syötteet, aloituskriteerit, tulosteet sekä mitä katselmoidaan vaiheen/aktiviteetin jälkeen eli välituotteet ja etapit.

1.2.2 Limittyminen ja palaaminen (3)

Prosessimallista tulisi ilmetä prosessin vaiheiden limittyminen/rinnakkaisuus sekä palaaminen aikaisempiin vaiheisiin.

1.2.3 Vaatimusmäärittelyt (4)

Vaatimusmäärittelyjen muuttumisen tulisi näkyä prosessimallista, miten vaatimuksia tuodaan prosessiin ja miten ne etenevät prosessin läpi. Mallista tulisi myös ilmetä takaisinkytkennät vaatimuksiin sekä mikä on vaatimusmäärittelyjen rooli prosessissa.

1.2.4 Resurssit (4)

Joidenkin mielestä työkalun rooli tulisi näkyä prosessimallista, koska siinä on tietyt suunnittelumenetelmät, jotka vaikuttaa prosessiin. Resurssien (ihmiset, työkalut, laitteet) mallintamisesta yleensä oli ristiriitaisia mielipiteitä: osan mielestä prosessin aktiviteettien suorittajien ja käytettyjen työkalujen mallintaminen ei ole oleellista.

1.2.5 Pehmeät, kommunikointi- / tietämysläheiset asiat (8)

Kommunikointiin ja tietämykseen liittyvät vaatimukset ovat pääsääntöisesti meidän johtamia haastatteluvastauksista. Haastatteluissa tietämyksen rooli ja tärkeys nousi esille esim. tehtyjen ratkaisujen jakaminen, tiedon/tietämyksen jakaminen yleensä, tieto siitä kuka tekee mitäkin jne.

Useimmat eivät osanneet sanoa kommunikoinnin mallintamisen tärkeydestä vaikkakin kommunikoinnin tärkeys ongelmien ratkaisussa ja tiedon jaossa nousi esille hyvinkin selkeästi. Kommunikoinnin tärkeys korostui myös uusilla työntekijöillä heidän tullessaan ryhmään oli kommunikoinnilla merkittävä rooli tehtäviin tutustuessa .

1.2.6 Prosessin rajapinnat (5)

Prosessimallista olisi hyvä näkyä osaprojektin ja SW(D)-ryhmän rajapinnat muihin (osa)projekteihin, ryhmiin sekä ulkopuolisiin, miten yhteydet toimivat, kumpaan suuntaan ja miten tieto kulkee. Lisäksi pitäisi näkyä, mitä tietoa ja missä yhteyksissä välitetään, mihin tietoa tarvitaan.

Rajapintaongelma tuotekehityksen ja tuotemarkkinoinnin välillä tuli ilmi etenkin asiakasvaatimusten yhteydessä, rajapinnan kuvaamista pidettiin toivottavana.

1.2.7 Yleisempiä sisältövaatimuksia (3)

Mallin tulisi kuvata millä tavalla ja missä järjestyksessä asioita tehdään, "Mitä tapahtuu ketjussa rautaan asti", mitä asioita pitää tehdä missäkin vaiheessa ja mitä käytäntöjä noudatetaan.

Appendix E: Topic Interview Report Sample

This appendix is a fraction of the interview report used as a basis for the initial modelling. This report was built by extracting the process information from the first draft interview reports such as the one in Appendix C.

Design phase

1 Level 1

1.1 Inputs (I) and Outputs (O)

- (I) E1 milestone document
- (I) Functional specifications
- (I) SW Architecture
- (I) TETRA standard
- (I) Documentation guide
- (I) Project plan
- (O) Module descriptions
- (O) E2 milestone documents

1.2 Entry (E) and Exit (Ex) criterion

- (E) E1 milestone
- (E) Functional specifications reviewed
- (E) Architecture reviewed
- (Ex) Module descriptions checked

1.3 Relationships to other activities

- Iteration between implementation and design
- Changes in customer requirements cause returning to the specification phase

1.4 Resources

–

1.5 Guidance / knowledge sharing / training

- Nokia/TETRA training from domain experts
- Project, group and team meetings

1.6 Interfaces

- DSP group
- Knowledge exchange with other units

2 Level 2

2.1 Creation of module description

2.2 Inputs (I) and Outputs (O)

- (I) Specifications
- (I) TETRA standard
- (I) Notes database
- (I) Interface specifications
- (I) SW architecture
- (O) Module descriptions

2.3 Entry (E) and Exit (Ex) criterion

- (E) SW architecture reviewed
- (Ex) Module description checked

2.4 Relationships to other activities

- Iteration between implementation
- Proceeding of standardisation can cause returning to module description
- Changes in customer requirements cause returning to the specification phase

2.5 Resources

–

2.6 Guidance / knowledge sharing / training

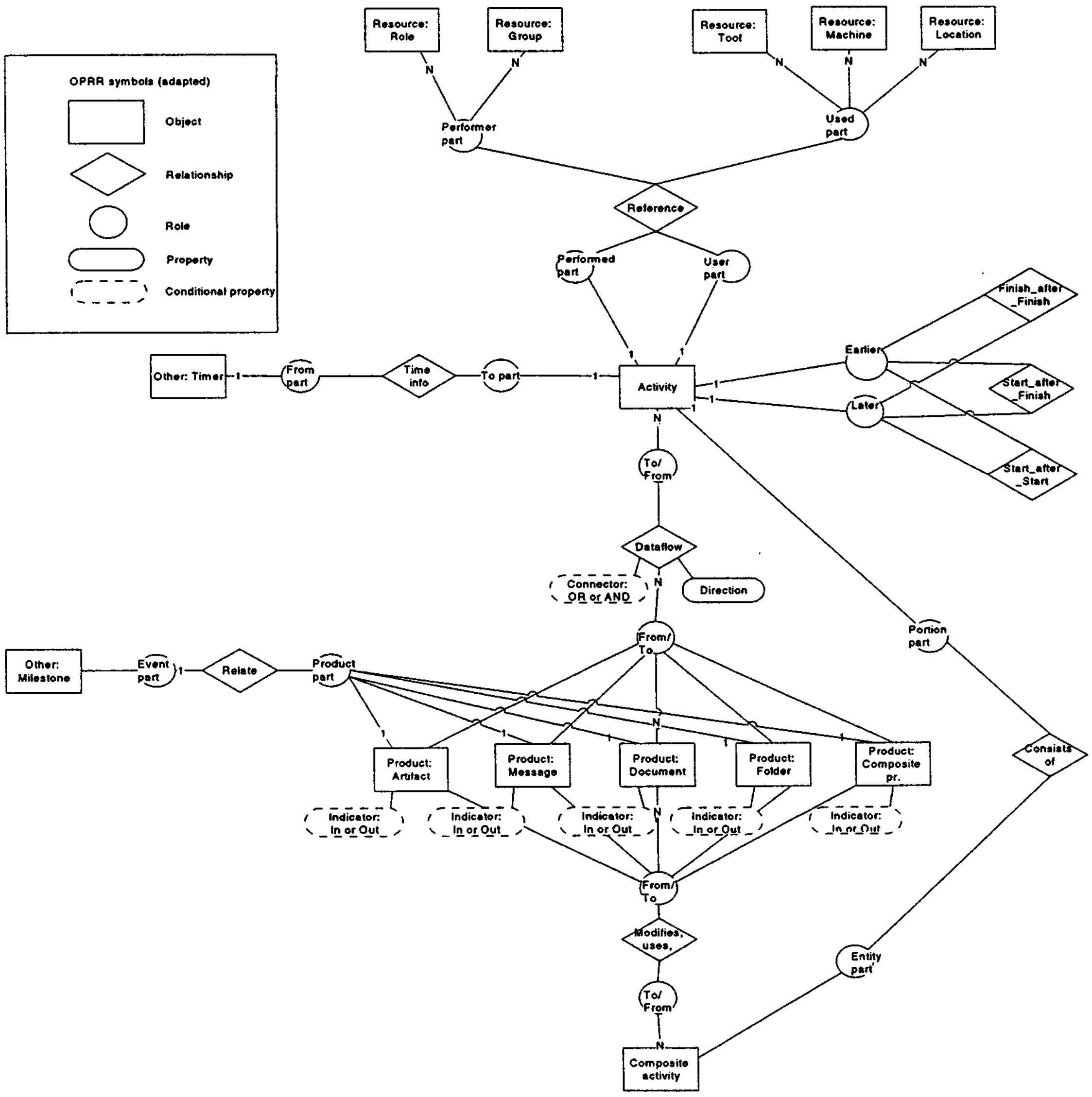
- Agree about module interfaces
- Nokia/TETRA experts

2.7 Interfaces

- Meetings and e-mail communication with other units about the interfaces and implementation

Appendix F: CONCEPTUAL (OPRR) MODEL OF VPML

The picture below offers a conceptual model of the initial PML which we selected as the basis for our PML engineering.



Appendix G: FINAL PML SYMBOL DESCRIPTION

This appendix contains the symbol description used in familiarising the actors with the PML. It covers all the symbols used, their semantics and some syntactical rules related to their use.

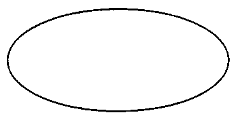
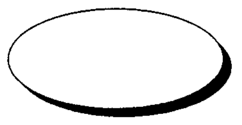
Textual description of the symbols used in the process model

The tables below present each of the symbols used with a picture, a name, symbol colour in the wall diagrams and a textual description.

1 Phases and activities (green)

Activities represent work that is performed in a process. Other constructs are used primarily to specify details about activities and to coordinate activities. Types of activities are leaf, and composite activities. Phases are a special case of composite activities and used only to describe the life cycle level of the software process.

Activities can be connected to one another with *Temporal connections*.

	*(leaf) Activity.* Work performed by one or more persons or groups.
	*Phase and composite activity.* Allows hierarchical composition of activities which means that a composite activity contains another graph in the next lower level of abstraction. In case of phase the symbol is stretched. Composite activities can have connections only to and from products.

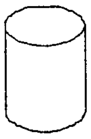
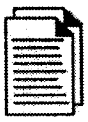
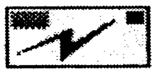
2 Products (blue)

Products represent items that are used, created, modified, and transferred among activities in a process.

Products are connected to activities with *Data flow* connections.

- A product on an outgoing connection from an activity represents a product that is created or modified by the activity.
- A product on an incoming data flow connection to an activity represents a product that is used to perform the activity.

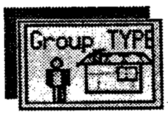
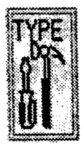
Products can be connected to one another with *Reference* connections. The connected products are then input to or output from the same activities.

	*Artifact.* A database or program source code. The symbol can also represent an executable program used or produced by an activity. "SW Module" and "Test cases" are examples.
	*Document.* One or more documents.
	*Message.* Bits of information that are output from or input to an activity, such as most commonly used "change request" or "guidance".

3 Resources

Resources are used to model real-world resources that are required to perform an activity.

Resources are connected to activities with *Reference* connections.

	*External group.* (light yellow) External group is a resource type which does not contain anything that belongs to the process modelled. It denotes interface to outside the process. An example would be "Vendor tool expert" when the group works in Telelogic (Customer support). External groups can be assigned to Leaf Activities.
	*Tool.* (brown) A kind of software or another tool needed to operate on a product in the process. "CASE tool" is an example of a Tool type that could be assigned to RU SW. Tool types can be assigned to any Activity.

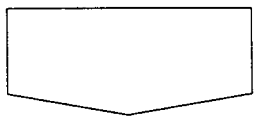
4 Connections

Connections are used to establish relationships between constructs and to pass product information between activities and help coordinate the scheduling of activities.

────────▶	*Data flow.* (black) A directed connection with an arrow indicating the direction in which information flows in the process graph. Data flow connections model short-term access to products that change each time an activity is performed.
— — — — -	*Reference.* (dark grey) Connects a resource to an activity. A Reference connection can also be

	used to connect products to one another. The connected products are then input to or output from the same activities.
- - - - - - - - - - - - - - - - -▶	*Temporal.* (light grey) A Temporal connection is specified with a name tag stating one of the following. Start_after_Finish connection from one activity to another indicates that the first activity must be finished before the second activity can begin. Start_after_Start connections are used to model two activities when one may not begin until the other has begun. The category of temporal connections can be extended if need be.
(AND)	*AND connector* (white). Combines connections into one.
(OR)	*OR connector.* (white) Separates a connection into many.

5 Other symbols

●⟶	*Output* indicator. (black) Attribute to an output product which is an output from the process. An *Output* indicator is connected to a product by placing it beside the product.
⟶●	*Input* indicator. (black) Attribute to an input product which is an input into the process from outside the process. An *Input* indicator is connected to a product by placing it beside the product.
	*Milestone / Review / Inspection.* (yellow) Used to denote a significant event in the process. "SW Architecture Review" and "Milestone E1 review" are examples. *Milestones* are connected to products with *Reference connections.*
	Meeting. (pink) An occasion where a number of people gather together to exchange information. Examples would be "Training" and "Group meeting". A *Meeting* can be connected to activities with *Data flow* connections. This stresses the information content of a *Meeting*. A *Meeting* can therefore be treated like any product. Another way of connecting a *Meeting* is with *Reference* connections. This merely tells that the *Meeting* is related to the execution of an activity. The two connections can be intermixed, so that one *Meeting* can be connected to different activities with different connections. Some connection(s) ought to be used.

?	*Condition* (if). (white) Precondition of an activity. For example: "*If* a new employee arrives, arrange training." A *Condition* is connected to an activity with a *Temporal connection*.
!	*Comment* (white). A textual remark. A *Comment* is not connected to any other symbol.

Appendix H: RESULTANT PROCESS MODEL

The pictures in this appendix provide an overview of the electronic version of the software process model created in our modelling case. The reader should not be bothered by the poor quality of the pictures. This is partly intentional to protect confidential data.

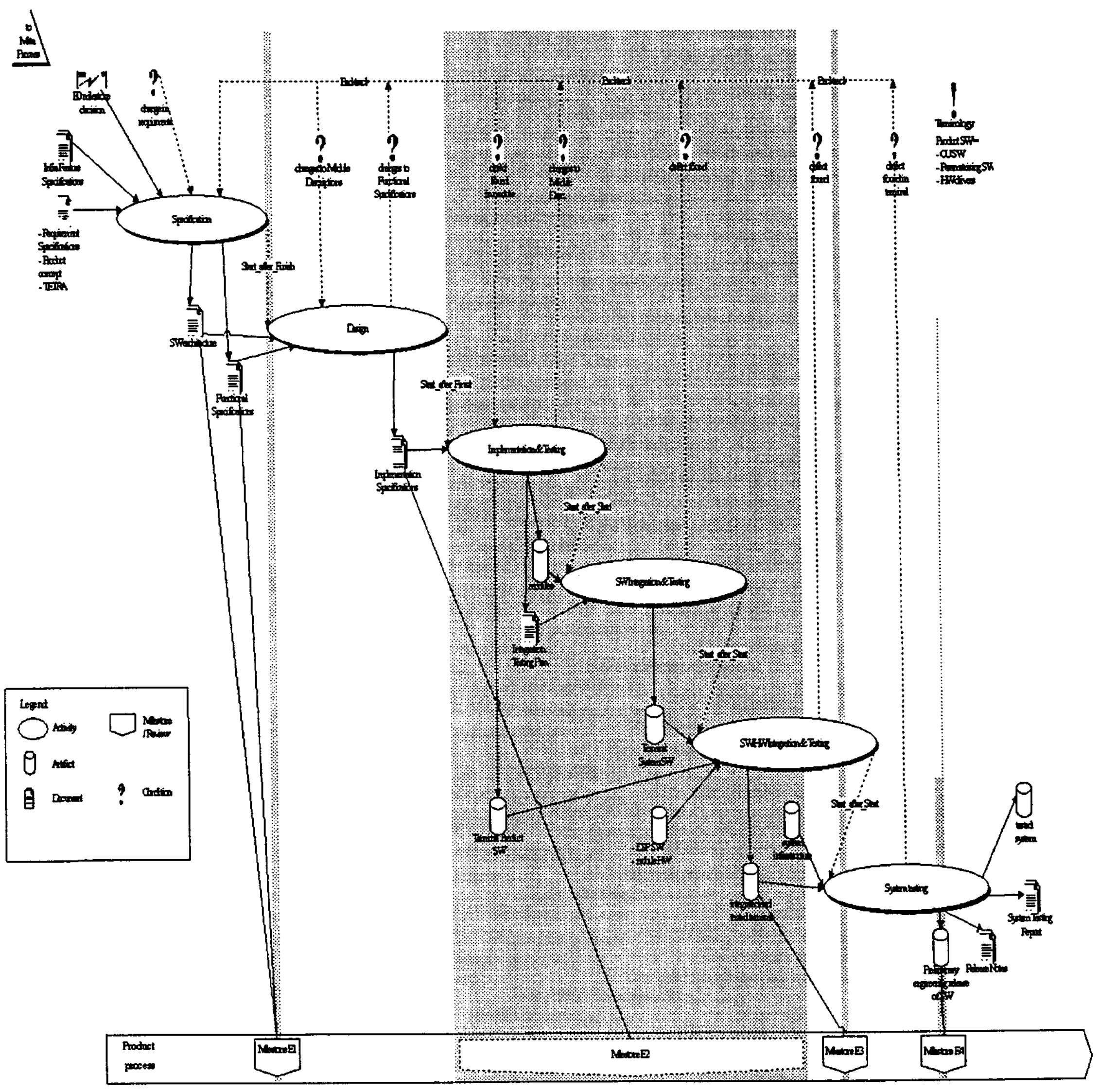


Figure 11. Life-Cycle View of Software Process

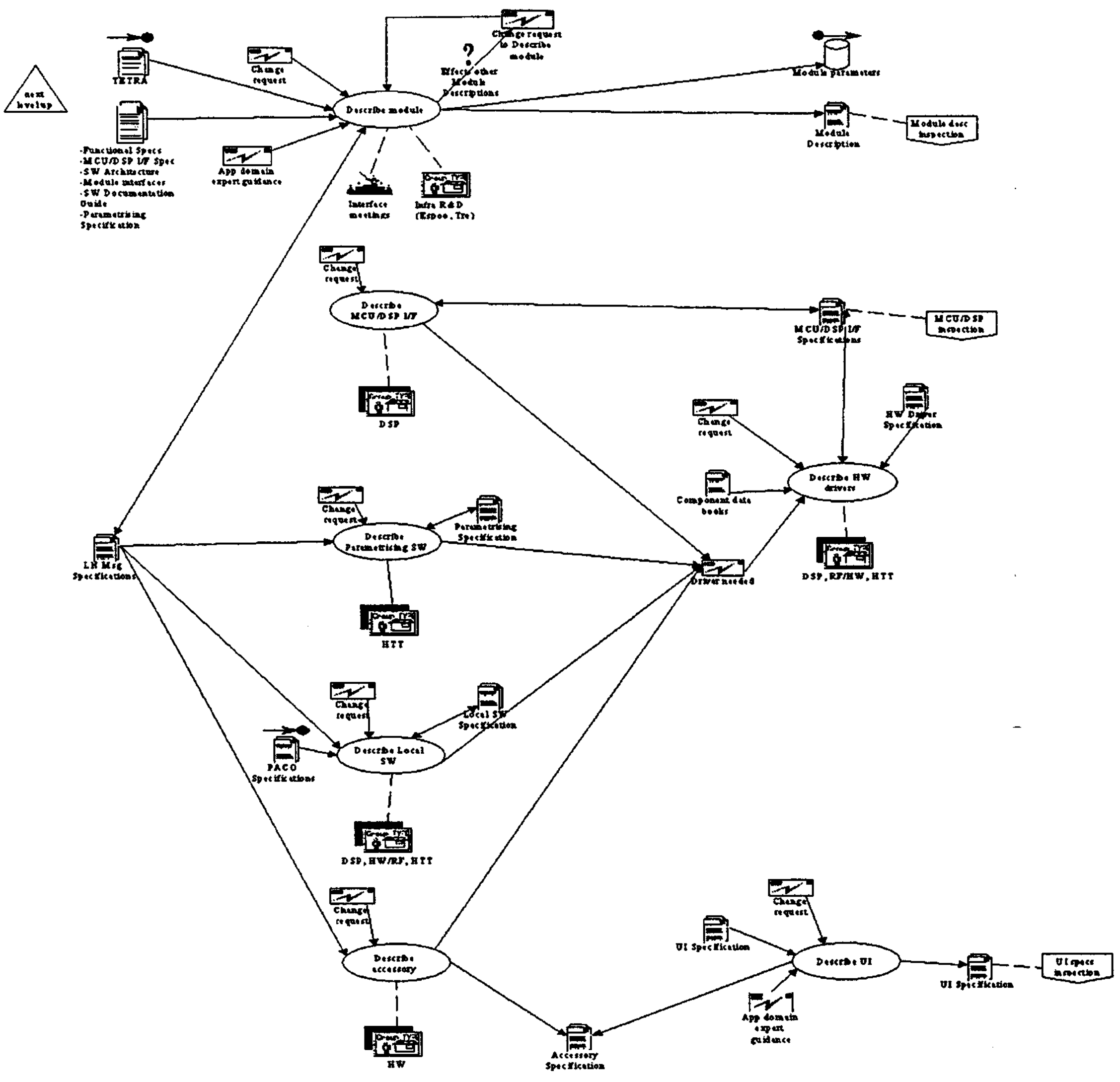


Figure 12. Design Phase of the Life-Cycle

Appendix I: WALL DIAGRAMS

The pictures in this appendix provide an overview of the wall diagram technique version of the software process model created in our modelling case. The reader should not be bothered by the poor quality of the pictures. This is partly intentional to protect confidential data.

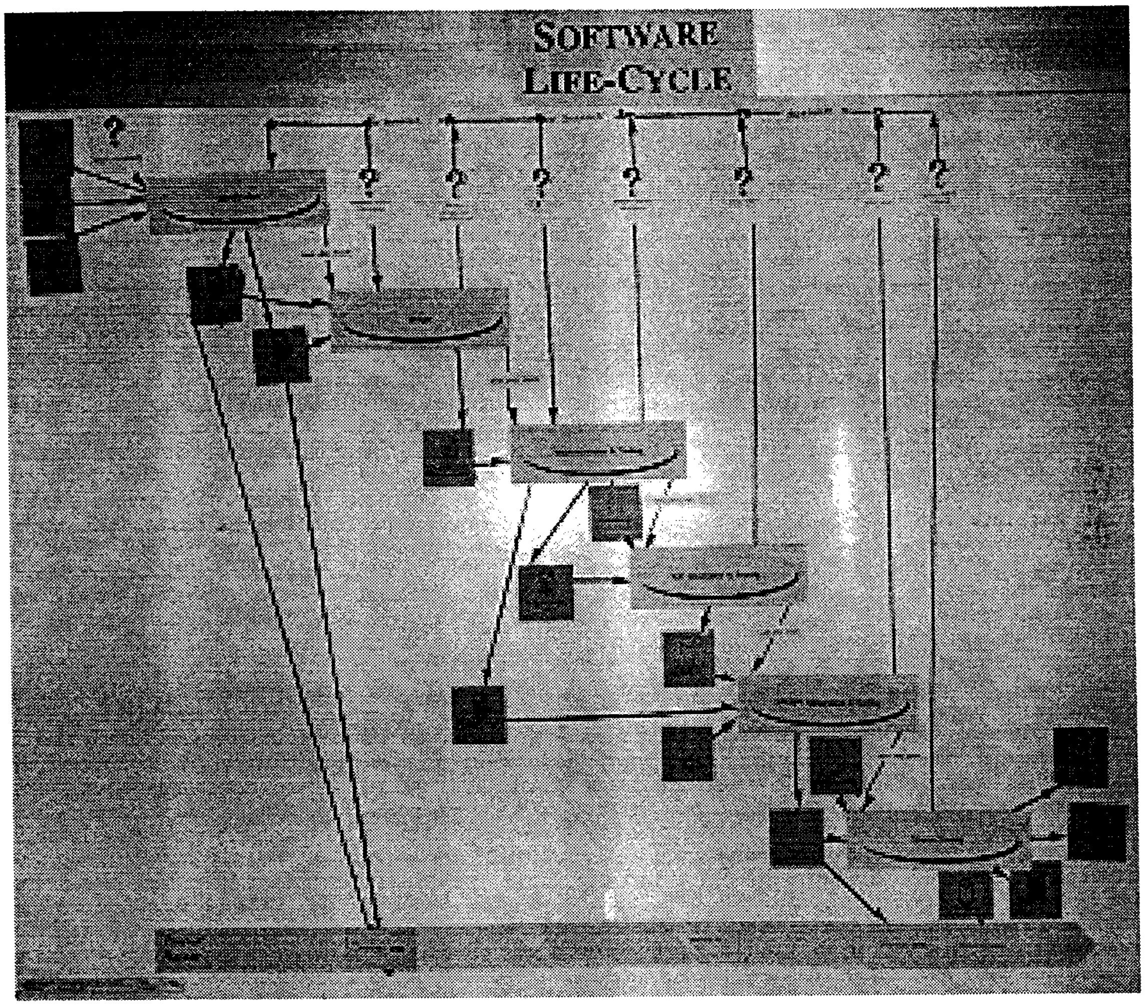


Figure 13. Life-Cycle View of Software Process

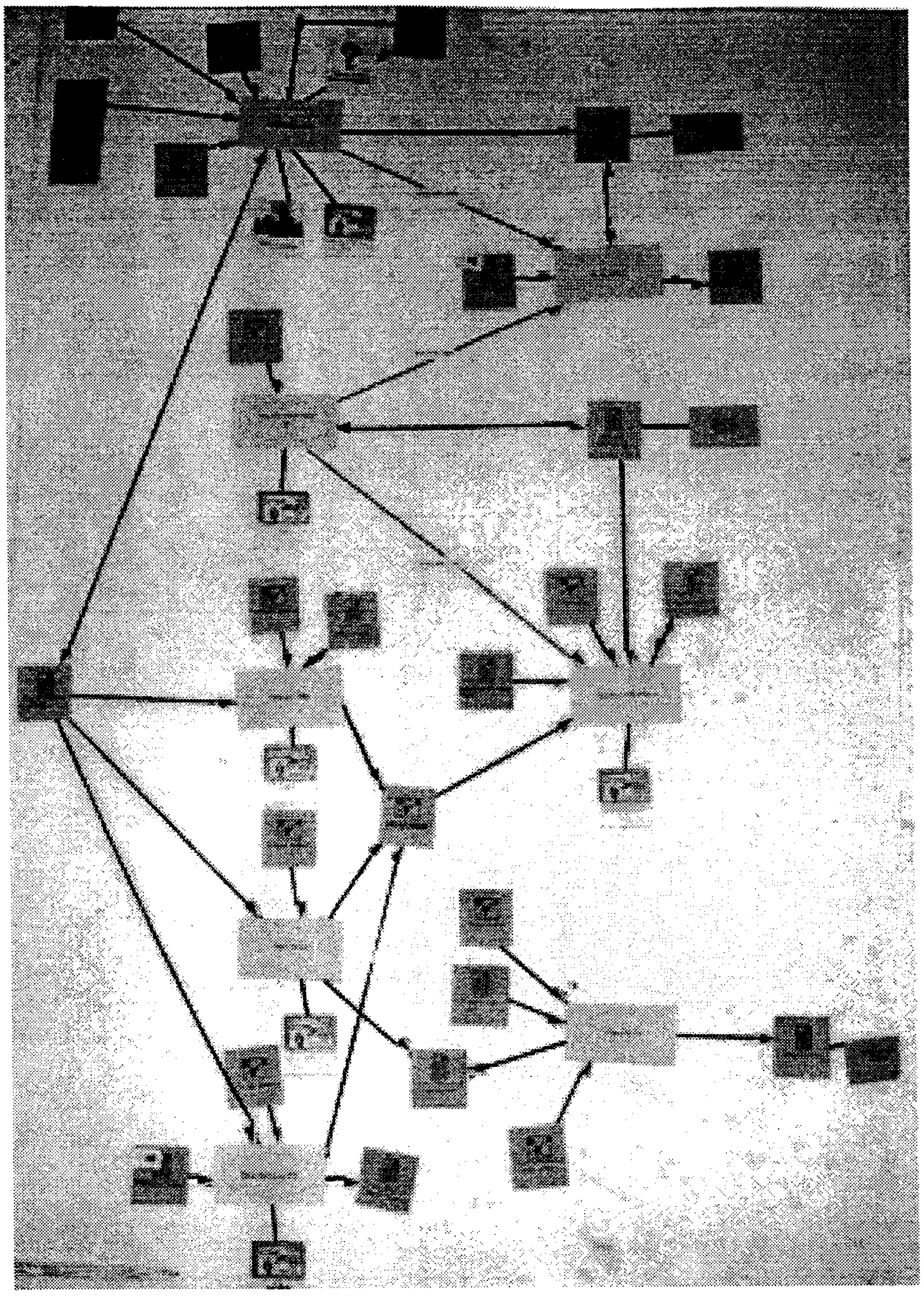


Figure 14. Design Phase of the Life-Cycle

Appendix J: TEXTUAL PROCESS DESCRIPTION TEMPLATE

This final appendix presents the initial template we constructed to complement the graphical notation of our PML.

Introduction

This description works as a template and an example of the textual description of the software process. The example contains a description of the specification phase including its activities.

The textual description of the phase is structured as follows:

Item	Description
Phase	process element of software life cycle
Level	level of granularity (e.g. life cycle, software process)
Purpose	short description of the phase
Entry criteria	input(s) which initiate the phase, including the state and origin of the input(s) (e.g. approved and TUMA)
Other inputs	inputs of the phase which doesn't initiate the phase
Exit criteria	output(s) of the phase, including the state and destination of the output(s)
Relationship to other phases	description of the relationships between activities
Activities	list of phase activities

The textual description of the activity is structured as follows:

Item	Description
(Composite) activity	process element of software process or meta level
Level	(refer to Phase above)
Purpose	(refer to Phase above)
Entry criteria	(refer to Phase above)
Other inputs	(refer to Phase above)
Exit criteria	(refer to Phase above)

Other outputs	outputs of the activity which doesn't close the activity
Relationship to other phases	(refer to Phase above)
External relationships	a resource external of the process (e.g. DSP, TUMA, Tool vendor)
Tool	a software or another tool resource needed to operate on a product in the process
Notice	extra information of the activity

Definitions:

State	Description
Proposal	draft version of the input or output
Approved	checked or reviewed version of the input or output

Origin/destination	Description
Internal	internal source or target of the phase

1 Phase: Design

1.1 Level

Life-cycle.

1.2 Purpose

To decompose the specifications and software architecture constructed in the previous phase into the implementation specifications which describe how the software is to be implemented. This include such activities as the writing of module descriptions and specifying MCU/DSP interface.

1.3 Entry criteria

Inputs	State	Origin
Functional specifications	approved	E1 milestone, Specification
SW architecture	approved	E1 milestone, Specification

1.4 Other inputs

None

1.5 Exit criteria

Outputs	State	Destination
Implementation specifications	proposal	E2 milestone, Implementation & Testing

1.6 Relationships to other phases

Design phase is the first phase of the software process and it has to end before next phase, Implementation and Testing, can start. There is a return loop to the specification from this phase when the requirements for the software changes. In addition, there is a return loop to this phase from the later phases when there is a need to change implementation specifications.

1.7 Activities

Nr	Activity
1.	Describe module
2.	Describe MCU/DSP interface
3.	Describe parametrising SW
4.	Describe local SW
5.	Describe accessory
6.	Describe HW drivers
7.	Describe UI

1.7.1 Activity: Describe module

1.7.1.1 Purpose

To write the module description which include interfaces to the other modules, parameters of these interfaces and data types. In addition, the module description may include the use of MSC charts and state transition diagrams to aid the designing work. The MSCs are used to describe various scenarios in which module may be involved and state transition diagrams are used to describe the various states in which module may be.

1.7.1.2 Entry criteria

Inputs	State	Origin
Functional specifications	approved	Specification

MCU/DSP interface specification	proposal	Specification
SW architecture	approved	Specification
Module interfaces	proposal	Internal

1.7.1.3 Other inputs

Inputs	State	Origin
SW documentation guide		Process support
Parametrising specification	approved	Specification
LN msg specification	proposal	Specification
TETRA standard		

1.7.1.4 Exit criteria

Outputs	State	Destination
Module description	proposal	Implementation and Testing

1.7.1.5 Other outputs

Outputs	State	Destination
Module parameters	proposal	HTT
LN msg specification	proposal	Internal
Change request		Internal

1.7.1.6 Relationships to other activities

No direct relationships between other activities in design phase.

1.7.1.7 External relationships

External	Description of the relationship
Infrastructure R & D	implementation related meetings and communication

1.7.1.8 Tools

Regular editors and drawing tools are used when appropriate.